Ignite-UX Custom Configuration Files

Table of Contents

Abstract

Ignite-UX for HP-UX addresses the need for system administrators to perform operating system installations, deployment, and recovery, often on a large scale. It provides the means for creating and reusing standard operating system configurations. Additionally, Ignite-UX delivers the ability to archive operating system configurations, and to use these archives to replicate systems, with the added benefit of speeding up the process. Ignite-UX also permits various customizations, and is capable of both interactive and unattended operating modes.

Introduction

The *Ignite-UX Administration Guide* explains how to use the Ignite-UX product. However, it does not cover in detail the configuration files used by Ignite-UX (and custom configuration files that you can write).

This white paper supplements the *Ignite-UX Administration Guide* with information to help you to understand what specific configuration files are attempting to accomplish, which makes creating and writing custom configuration files easier. Readers are assumed to understand how to install a system using Ignite-UX. The information in this white paper is about managing, writing, and modifying configuration files.

In developing this white paper, Ignite-UX versions from B.5.1.*x* to C.6.0.*x* were used.

Note:
Care was taken to prevent unintended line wraps from occurring in the examples in this document; however, due to formatting limitations, some might exist. This may affect the usefulness of examples if you use them without verifying the syntax using `instl_adm` with the `-T` option.

Typographic Conventions

The following typographical conventions are used throughout this white paper.

audit(5)

An HP-UX manpage. "audit" is the name and "5" is the section in the *HP-UX Reference*. On the Web and on the Instant Information media, it may be a hot link to the manpage itself. From the HP-UX command line, enter "man audit" or "man 5 audit" to view the manpage. For more information, refer to *man*(1).

Book Title

The title of a book. On the Web and on the Instant Information media, it may be a hot link to the book itself.

Emphasis

Text that is emphasized.

Emphasis

Text that is strongly emphasized.

`ComputerOut`

Text displayed by the computer.

`Command`

A command name or qualified command phrase.

`Computer`

Computer font indicates literal items displayed by the computer. For example:

```
file not found
```

`Filename`

Text that shows a filename and/or filepath.

`UserInput`

Commands and other text that you type.

Variable

The name of a variable that you may replace in a command or function or information in a display that represents several possible values.

`[ ]`

The contents are optional in formats and command descriptions.

`{ }`

The contents are required in formats and command descriptions. If the contents are in a list separated by |, you must choose one of the items

. . .

The preceding element may be repeated an arbitrary number of times.

|

Separates items in a list of choices.

<*MAC*>

When used as part of a file name, this placeholder variable refers to the Media Access Control (MAC) address of the Ignite-UX client system under `/var/opt/ignite/clients`. This may also be referred to as Link-Level Address (LLA).

<*DATE-TIME*>

This indicates a directory created by Ignite-UX that has a fixed format containing the date and time it was created, for example "`2004-01-12,15:23`".

HP-UX 11i release names and release identifiers

Each HP-UX 11i release has an associated release name and release identifier. Table 1 shows the releases available for HP-UX 11i.

Table 1

Release Name	Release Identifier	Supported Processor Architecture
HP-UX 11i v1	B.11.11	PA-RISC
HP-UX 11i v1.5	B.11.20	Intel® Itanium
HP-UX 11i v1.6	B.11.22	Intel® Itanium
HP-UX 11i v2	B.11.23	Intel® Itanium PA-RISC[1]

The *uname*(1) command with the `-r` option returns the release identifier.

You can also determine the update release date and the Operating Environment by entering the following:

```
# swlist | grep HPUX11i
```

The resulting output lists the current release identifier, update release date, and Operating Environment. For example:

```
HPUX11i-TCOE     B.11.23.0409 HP-UX Technical
 Computing Operating Environment Component
```

The preceding revision string represents the following:

```
B.11.23 = HP-UX 11i v2

0409 = September 2004 Update Release
```

1 PA-RISC is supported on HP-UX 11i v2 starting with the September 2004 release.

Configuration files and **INDEX** files

This section discusses the contents of configuration files and INDEX files to help you understand the purpose of the files, not to document what is occurring in the files.

The **INDEX** file

The file /var/opt/ignite/INDEX is used during cold-installations. The following example only has one cfg clause; usually there are more. Each cfg clause defines a series of configuration files, along with a description, that together form a cohesive configuration that is used to install a system.

```
$ pwd
/var/opt/ignite
$ cat INDEX
# /var/opt/ignite/INDEX
# This file is used to define the Ignite-UX configurations
# and to define which config files are associated with each
# configuration.  See the ignite(5), instl_adm(4), and
# manage_index(1M) man pages for details.
#
# NOTE: The manage_index command is used to maintain this file.
#       Comments, logic expressions and formatting changes are not
#       preserved by manage_index.
#
# WARNING: User comments (lines beginning with '#' ), and any user
#          formatting in the body of this file are _not_ preserved
#          when the version of Ignite-UX is updated.
#
cfg "HP-UX B.11.11 Default" {
        description "This selection supplies the default system configuration
that HP supplies for the B.11.11 release."
        "/opt/ignite/data/Rel_B.11.11/config"
        "/opt/ignite/data/Rel_B.11.11/hw_patches_cfg"
        "/var/opt/ignite/config.local"
}
```

If the cfg clause in the INDEX file were to be the default cfg clause, in a non-interactive installation this would cause the cfg clause to be selected for installation by default, it would look like the following:

```
cfg "HP-UX B.11.11 Default" {
        description "This selection supplies the default system configuration
that HP supplies for the B.11.11 release."
        "/opt/ignite/data/Rel_B.11.11/config"
        "/opt/ignite/data/Rel_B.11.11/hw_patches_cfg"
```

```
        "/var/opt/ignite/config.local"
}=TRUE
```

Of course, the `INDEX` file is never to be edited manually; instead, the `manage_index` command should be used to maintain this file. The actual command needed to set a `cfg` clause to be the default clause is as follows:

```
# manage_index -e -c "HP-UX B.11.11 Default"
```

The `manage_index` command operates on the `/var/opt/ignite/INDEX` file by default so the `INDEX` file does not need to be specified on the command line.

The **`CINDEX`** file

The `CINDEX` file is very different from the `INDEX` file. The `CINDEX` file is created and managed by the `make_net_recovery` command. The format of the `CINDEX` file is the same as the `INDEX` file. For example:

```
# cat /var/opt/ignite/clients/<MAC>/CINDEX
# This file is used to define the Ignite-UX configurations
# and to define which config files are associated with each
# configuration.  See the ignite(5), instl_adm(4), and
# manage_index(1M) man pages for details.
#
# NOTE: The manage_index command is used to maintain this file.
#       Comments, logic expressions and formatting changes are not
#       preserved by manage_index.
#
# WARNING: User comments (lines beginning with '#' ), and any user
#          formatting in the body of this file are _not_ preserved
#          when the version of Ignite-UX is updated.
#
cfg "2003-10-08,12:41 Recovery Archive" {
        description "Recovery Archive"
        "recovery/2003-10-08,12:41/system_cfg"
        "recovery/2003-10-08,12:41/control_cfg"
        "recovery/2003-10-08,12:41/archive_cfg"
}
cfg "2003-10-08,12:45 Recovery Archive" {
        description "Recovery Archive"
        "recovery/2003-10-08,12:45/system_cfg"
        "recovery/2003-10-08,12:45/control_cfg"
        "recovery/2003-10-08,12:45/archive_cfg"
}=TRUE
```

This file contains the `cfg` clauses needed to recover a system from a network recovery archive. The `make_net_recovery` command automatically manages the `CINDEX` file so

older information is removed[2] and the latest recovery archive is set as the default archive that is recovered.

The per-client configuration file

The configuration file that can exist in a per-client directory (`var/opt/ignite/clients/<MAC>`) overrides all other files that an Ignite-UX client may use. That is, any configuration values set in this file override all other files with one exception that is discussed elsewhere (see "Order of precedence of configuration files" for more information). During the installation process, Ignite-UX saves to this file the configuration information that was used, which facilitates repeated installations of the same system. If this file exists, it typically selects a `cfg` clause from the `CINDEX` or `INDEX` files. For example:

```
# cat config
cfg "2003-10-08,12:45 Recovery Archive"=TRUE
_hp_cfg_detail_level="ipvs"
#
# Variable assignments
#
init _hp_root_disk="8/16/5.6.0"
init _hp_root_grp_disks=1
init _hp_root_grp_striped="NO"
...
```

In this case, a recovery archive has been selected. (This also indicates that a recovery archive may have been recovered onto this system at some time in the past.)

The global **`config.local`** file

The file `/var/opt/ignite/config.local` is included by default into all new `cfg` clauses in the `/var/opt/ignite/INDEX` file. For this reason, configuration data stored in this file is used on all new installations.

The recovery **`config.local`** file

For `make_net_recovery`, if the file `/var/opt/ignite/clients/<MAC>/recovery/config.local` exists on the Ignite-UX server, the contents are automatically included into the configuration used by the client during a network recovery.

For `make_tape_recovery`, if the file `/var/opt/ignite/recovery/config.local` exists, the contents are automatically included into the configuration used by the client even though it is not explicitly referenced. The content of this file is placed into the LIF on the tape.

---

[2] For more information, refer to *`make_net_recovery`*`(1M)` and review the description of the `-n` option.

Note:
If you use the `-s` option with `make_tape_recovery`, the same `config.local` file used by `make_net_recovery` is used instead of `/var/opt/ignite/recovery/config.local`, as previously described.

Order of precedence of configuration files

Configuration files have an order of precedence when a client is reading configurations from an Ignite-UX server. The precedence is as follows:

1. The per-client configuration file (`/var/opt/ignite/clients/<MAC>/config`) — Any variable set in the per-client `config` file takes precedence over any variable set anywhere else. Using the per-client `config` file in the preceding example, the `cfg` clause `2003-10-08,12:45 Recovery Archive` in the `CINDEX` file is selected no matter what the `INDEX` or `CINDEX` have set to `TRUE`. (This, of course, you can change this using the Ignite-UX Graphical User Interface (GUI).)
2. The per-client `CINDEX` file (`/var/opt/ignite/clients/<MAC>/CINDEX`) — This file contains recovery archive configurations. Any `cfg` clause set to `TRUE` in this file automatically selects that `cfg` clause if there is no per-client `config` file.
3. The global `INDEX` file (`/var/opt/ignite/INDEX`)— This is the last file to have precedence. The default clause from the `INDEX` file is the one that is set to `TRUE`. (None of the `cfg` clauses in the `INDEX` file has been set to `TRUE`.)

Configuration also has an order of precedence when you are booting a system. The precedence is as follows:

1. The first 8 KB of the installation file system has the highest precedence since it is the first configuration read by Ignite-UX.
2. During media installations, if there is a LIF file called CUSTOM on any device, this file is loaded and parsed after the configuration from the installation file system has been processed. Parsing the configuration after loading all other configurations means that it has the highest order of precedence (when booting from a CD or tape)[3].

Testing the order of precedence

It is possible to test the order of precedence using the `instl_dbg` command:

```
# pwd
/var/opt/ignite/clients/<MAC>
# cat CINDEX
...
cfg "2003-10-08,12:41 Recovery Archive" {
        description "Recovery Archive"
        "recovery/2003-10-08,12:41/system_cfg"
        "recovery/2003-10-08,12:41/control_cfg"
```

[3] This feature is being considered for obsolescence. This part of the installation process is very time consuming for systems with a very large number of disks.

```
        "recovery/2003-10-08,12:41/archive_cfg"
}
cfg "2003-10-08,12:45 Recovery Archive" {
        description "Recovery Archive"
        "recovery/2003-10-08,12:45/system_cfg"
        "recovery/2003-10-08,12:45/control_cfg"
        "recovery/2003-10-08,12:45/archive_cfg"
}=TRUE
# manage_index -e -c "2003-10-08,12:41 Recovery Archive" \
> -i /var/opt/ignite/clients/box1/CINDEX
# cat CINDEX
...
cfg "2003-10-08,12:41 Recovery Archive" {
        description "Recovery Archive"
        "recovery/2003-10-08,12:41/system_cfg"
        "recovery/2003-10-08,12:41/control_cfg"
        "recovery/2003-10-08,12:41/archive_cfg"
}=TRUE
cfg "2003-10-08,12:45 Recovery Archive" {
        description "Recovery Archive"
        "recovery/2003-10-08,12:45/system_cfg"
        "recovery/2003-10-08,12:45/control_cfg"
        "recovery/2003-10-08,12:45/archive_cfg"
}
```

So far, the `CINDEX` for this client has been changed so that the older recovery archive is the default archive. This is needed in order to show how precedence works. In the `config` file, the following `cfg` clause is selected:

```
# cat config
cfg "2003-10-08,12:45 Recovery Archive"=TRUE
_hp_cfg_detail_level="ipvs"
...
```

You can now use the `instl_dbg` command to print out the fully parsed configuration (The `-D` option points it to a per-client directory containing the files it needs. The `-f` option designates where to write the parsed configuration). This will occur with and without the `config` file being present in the per-client directory:

```
# instl_dbg -D . -f /tmp/client.with.config
# mv config config.save
# instl_dbg -D . -f /tmp/client.with.no.config
```

With the per-client `config` file present, the `cfg` clause `2003-10-08,12:45 Recovery Archive` is selected:

```
# head /tmp/client.with.config
cfg "2003-10-08,12:45 Recovery Archive"=TRUE
server="10.0.0.3"
is_net_info_temporary=FALSE
init _hp_keyboard="PS2_DIN_US_English"
system_name="host"
ip_addr[]="10.0.0.1"
netmask[]="0xffffff00"
route_gateway[0]="10.0.0.2"
route_destination[0]="default"
_hp_cfg_detail_level="ibnpvcdsrlLtfh"
...
```

Without the per-client `config` file present, the `cfg` clause is selected by the `CINDEX` file (`2003-10-08,12:41 Recovery Archive`) instead:

```
# head /tmp/client.with.no.config
cfg "2003-10-08,12:41 Recovery Archive"=TRUE
server="10.0.0.3"
is_net_info_temporary=FALSE
init _hp_keyboard="PS2_DIN_US_English"
system_name="host"
ip_addr[]="10.0.0.1"
netmask[]="0xffffff00"
route_gateway[0]="10.0.0.2"
route_destination[0]="default"
_hp_cfg_detail_level="ibnpvcdsrlLtfh"
...
```

The `instl_dbg` command is covered in more detail in a later section.

What is in a configuration (**`cfg`**) clause?

The question now is, what is in a `cfg` clause? The following example shows a `cfg` clause taken from the `/var/opt/ignite/INDEX` file:

```
cfg "HP-UX B.11.11 Default" {
        description "This selection supplies the default system configuration
that HP supplies for the B.11.11 release."
        "/opt/ignite/data/Rel_B.11.11/config"
        "/opt/ignite/data/Rel_B.11.11/hw_patches_cfg"
```

```
        "/var/opt/ignite/config.local"
}=TRUE
```

The `cfg` clause needs the following information:

1. A name (for example, HP-UX B.11.11 Default)
2. A description (see the description keyword)
3. A list of one or more configuration files referenced by this cfg clause
4. One or more configuration files that define software that may be installed[4]

When a `cfg` clause is selected (it has been set to `TRUE` and nothing of higher precedence changes this), its configuration files are processed in order and evaluated. This turns the set of configuration files into something that a client can use to install itself.

The **`make_net_recovery`** configuration files

A `make_net_recovery` session always creates the same configuration files, and you can see what they are by looking at a `cfg` clause in a `CINDEX` file:

```
cfg "2003-10-08,12:45 Recovery Archive" {
        description "Recovery Archive"
        "recovery/2003-10-08,12:45/system_cfg"
        "recovery/2003-10-08,12:45/control_cfg"
        "recovery/2003-10-08,12:45/archive_cfg"
}
```

The three configuration files are as follows:

1. `system_cfg` –-
   This file is produced by the `save_config` command. It contains the file system layout, networking information, and hardware instance numbers for the system.
2. `control_cfg` –
   This file contains definitions of Ignite-UX variables and the commands needed to import volume groups back into the final system.
3. `archive_cfg` –
   This file contains the software definition of the archive that is used for recovery (including impacts keywords, etc.).

The `CINDEX` file provides a higher level of precedence than the `INDEX` file, so any `cfg` clause selected in the `CINDEX` file is used in preference to anything in the `INDEX` file in a non-interactive installation for a system. In an interactive installation, the recovery archive is automatically selected for installation so you can manually select another configuration for installation on the **Basic** tab in the Ignite-UX GUI.

---

[4] This `cfg` clause is a default clause set up by Ignite-UX. This clause, as shown, does not have any configuration files that define software to be installed. Therefore, it is incomplete and cannot be used to install a system.

The **make_tape_recovery** configuration files

The make_tape_recovery command still creates the same three configuration files that make_net_recovery does. For example:

```
# pwd
/var/opt/ignite/recovery/2004-01-12,15:23
# ll *cfg
-rw-r--r--   1 root       sys            3032 Jan 12 15:24 archive_cfg
-rw-r--r--   1 root       sys             963 Jan 12 15:24 control_cfg
-rw-r--r--   1 root       sys            7223 Jan 12 15:24 system_cfg
```

The main difference is that the three configuration files end up in a single LIF file on the tape when the recovery tape is created. The order of precedence here is unimportant since the tape is the only device used and all of the configuration files are concatenated together into one LIF file (CONFIG).

Files created by **make_config**

The make_config command is used to create configuration files that provide enough information to allow the installation of software bundles from Software Distributor (SD) depots. The make_config command is covered in a later section.

Using the **manage_index** command

A variety of uses

The manage_index command has many different options. This section describes every form the command can take, as well as its uses.

You should never manually maintain INDEX files. Instead, you should always use manage_index to maintain them. The manage_index command does not maintain any formatting or comments that may have been added to index files by an Ignite-UX administrator.

The following are all the forms defined in *manage_index*(1M):

```
manage_index -a -f config_filename [-c cfg_clause_name| -r release]
     [-p] [-v] [-i index_filename]

manage_index -a -s script_file_name [-p] [-v] [-i index_filename]

manage_index -d -c cfg_clause_name | -r release [-p] [-v]
     [-i index_filename]

manage_index -e -c cfg_clause_name [-p] [-v] [-i index_filename]
```

```
manage_index -l -c cfg_clause_name | -r release ] [-o] [-v]
     [-i index_filename]

manage_index -l -o [-v] [-i index_filename]

manage_index -m old_clause_name -c new_clause_name [-p] [-v]
     [-i index_filename]

manage_index -n existing_clause_name -c new_clause_name [-p] [-v]
     [-i index_filename]

manage_index -t -f config_file_name [-c cfg_clause_name| -r release]
     [-p] [-v] [-i index_filename]

manage_index -t -s script_file_name [-p] [-v] [-i index_filename]

manage_index -w -c cfg_clause_name [-v] [-i index_filename]

manage_index -x -c cfg_clause_name [-v] [-i index_filename]
```

Note:
The examples presented are for illustration purposes only; they show the intended purpose of the command and use fictional configuration files, index files, and scripts.

Although the `-p` (preview, make no changes) and `-v` (verbose) options are available to most or all forms of `manage_index`, they are not discussed with any of the examples.

Adding a configuration file to a clause or "release"

When you need to add a configuration file to a clause or release you need to use the `-a` option. The `-i` option must be given a fully qualified path if it is used (it cannot be relative).

```
manage_index -a -f config_filename [-c cfg_clause_name| -r release]
   [-p] [-v] [-i index_filename]
```

Neither the `-c` nor `-r` options are required. If you use the `-c` option, you can explicitly name the configuration clause to which to add the configuration file. If you use the `-r` option to give an
HP-UX release identifier (for example, `B.11.23`), the configuration file will be added to each configuration clause that satisfies one of the following conditions:

- One of the configuration files in a `cfg` clause defines the release keyword with a value that matches the release identifier given with the `-r` option.
- One of the configuration files in a `cfg` clause starts with the path `/var/opt/ignite/data/Rel_`, and the value of the release identifier immediately after it matches the release identifier given with the `-r` option.
- One of the configuration files in a `cfg` clause starts with the path `/opt/ignite/data/Rel_`, and the value of the release identifier immediately after it matches the release identifier given with the `-r` option.

Without the `-c` or `-r` option, the value of the release identifier is determined in the same way that clauses are matched with the `-r` option. The following, in order, is performed to try to determine the release identifier that will be used:

- The configuration file defines the release keyword, and the value of the release keyword provides the value for the release identifier.
- The configuration file starts with the path `/var/opt/ignite/data/Rel_`, and the value of the release identifier from this path provides the value for the release identifier.
- The configuration file starts with the path `/opt/ignite/data/Rel_`, and the value of the release identifier from this path provides the value for the release identifier.

Once a release identifier has been determined for the configuration file, the `manage_index` command behaves as though it was given that release identifier with the `-r` option.[5]

In the following example, if you subsequently tried to add a configuration file based upon a release, it would not work because none of the configuration files contains an operating system release in their path names. However, you can add a file based upon a release if you first add the expected information in the path.

```
$ cat INDEX
cfg "testing" {
        description "testing clause"
}
$ touch config_a config_b
$ print "release=B.11.11" > config_c
$ manage_index -a -f /var/tmp/config_c -c "testing" \
> -i /var/tmp/INDEX
$ cat INDEX
...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
}
```

[5] This information is applicable to all forms of the `manage_index` command that need to search for or match release identifiers.

The following example adds the configuration file `/opt/ignite/data/Rel_B.11.11/config` to the `cfg` clause that then enables you to add the configuration file `/var/tmp/config_a`.

```
manage_index -a -f /opt/ignite/data/Rel_B.11.11/config -c "testing" \
> -i /var/tmp/INDEX
$ manage_index -a -f /var/tmp/config_a -r B.11.11 \
> -i /var/tmp/INDEX
$ cat INDEX
...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
```

Adding scripts to the **INDEX** file

As part of the syntax for configuration index files, there is a keyword called `scripts`, which enables you to define user-selectable scripts that can be run during installation and recovery:

```
manage_index -a -s script_file_name [-p] [-v] [-i index_filename]
```

The `manage_index` command with the `-a` and `-s` options allows you to add scripts into an `INDEX` file. For example:

```
$ manage_index -a -s /var/tmp/script_a \
> -i /var/tmp/INDEX
$ cat INDEX
…
scripts {
        "/var/tmp/script_a"
}
```

On the **Advanced** tab in the Ignite-UX GUI, the sample script appears so it can be selected and run by a user. The `scripts` clause in the `INDEX` file is global in scope; scripts defined here are available for selection with any `cfg` clause in the `INDEX` file. You should be careful not to define scripts that make assumptions about the release they will be run on or are not portable across all HP-UX releases.

Removing **cfg** clauses from an **INDEX** file

The `-d` option is used to remove `cfg` clauses from an `INDEX` file using `manage_index`:

```
manage_index -d -c cfg_clause_name | -r release [-p] [-v]
    [-i index_filename]
```

The following example removes the `cfg` clause `"testing two"` from the `INDEX` file completely:

```
$ cat INDEX
...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
$ manage_index -d -c "testing two" -i /var/tmp/test/INDEX
```

You can see that the `cfg` clause `"testing two"` has been removed from the `INDEX` file:

```
$ cat INDEX...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
```

The `-r` option operates on all `cfg` clauses applicable to a release identifier. If you reference `cfg` clauses by release, you need to be careful. In the following example, both `cfg` clauses have a release identifier that matches `B.11.11`, so both are removed from the `INDEX` file.

```
$ cat INDEX.save
...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
```

```
        "/var/tmp/config_a"
}
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
$ manage_index -d -r B.11.11 -i /var/tmp/test/INDEX
$ cat INDEX
...
```

Setting the default **cfg** clause in an **INDEX** file

Using the following command, you can easily set which of the `cfg` clauses in an `INDEX` file is the default `cfg` clause:

```
manage_index -e -c cfg_clause_name [-p] [-v] [-i index_filename]
```

The following example changes the default `cfg` clause between two different `cfg` clauses:

```
$ cat INDEX
...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
$ manage_index -e -c "testing" -i /var/tmp/INDEX
$ cat INDEX
...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
```

```
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}=TRUE
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
$ manage_index -e -c "testing two" -i /var/tmp/INDEX
$ cat INDEX
...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}=TRUE
```

Listing the names of **cfg** clauses in an **INDEX** file

The following syntax for the `manage_index` command lists the names of all of the `cfg` clauses in an `INDEX` file:

```
manage_index -l -c cfg_clause_name | -r release ] [-o] [-v]
    [-i index_filename]
```

For example:

```
$ cat INDEX
...
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
```

```
        "/var/tmp/config_a"
}
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}=TRUE
$  manage_index -l -i /var/tmp/INDEX
testing
testing two
```

Being able to list the `cfg` clauses also makes it easier to script Ignite-UX tasks. An example of this could be a sanity-checking script that ensures all of the configuration files referenced from within `cfg` clauses exist and are readable.[6] Such a script might look like:

```
#!/usr/bin/sh
/opt/ignite/bin/manage_index -l | while read CFG
do
{
eval /opt/ignite/bin/manage_index -w -c \"${CFG}\" | while read FILE
do
if [[ ! -f $FILE ]]
then
print -u2 "Error: file $FILE does not exist"
fi
if [[ ! -r $FILE ]]
then
print -u2 "Error: file $FILE is not readable"
fi
done
}
done"
```

Listing the name of the default **`cfg`** clause in an **`INDEX`** file

This form of the `manage_index` command enables you to see which `cfg` clause is the current default `cfg` clause:

```
manage_index -l -o [-v] [-i index_filename]
```

For example:

```
$ cat INDEX
...
cfg "testing" {
```

[6] Note that this script is a trivial example. The `instl_adm` command with the `-T` option would perform these tasks as well as sanity-check the syntax within the configuration files.

```
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}=TRUE
$ manage_index -l -o -i /var/tmp/INDEX
Default config clause is "testing two"
        in index file: /var/tmp/INDEX
```

Renaming a **cfg** clause in an **INDEX** file

The following form of `manage_index` enables you to rename a `cfg` clause. (This may be useful when you need to swap two clauses.)

```
manage_index -m old_clause_name -c new_clause_name [-p] [-v]
    [-i index_filename]
```

The following example swaps two `cfg` clauses:

```
$ manage_index -l -i /var/tmp/INDEX
testing
testing two
$ manage_index -m "testing two" -c "t two" -i /var/tmp/INDEX
$ manage_index -l -i /var/tmp/INDEX
testing
t two
$ manage_index -m "testing" -c "testing two" -i /var/tmp/INDEX
$ manage_index -l -i /var/tmp/INDEX
testing two
t two
$ manage_index -m "t two" -c "testing" -i /var/tmp/INDEX
$ manage_index -l -i /var/tmp/INDEX
testing two
testing
```

Creating a new **cfg** clause from an existing clause

Using the following form of `manage_index`, you can easily to create a new `cfg` clause from an existing clause:

```
manage_index -n existing_clause_name -c new_clause_name [-p] [-v]
    [-i index_filename]
```

For example:

```
$ manage_index -n "testing" -c "old testing two" -i /var/tmp/INDEX
$ manage_index -l -i /var/tmp/INDEX
testing two
testing
old testing two
```

Removing a configuration file from a **cfg** clause

Earlier you learned how to add a configuration file to a clause. Now you can do the opposite, which is to remove configuration files from `cfg` clauses:

```
manage_index -t -f config_file_name [-c cfg_clause_name | -r release]
    [-p] [-v] [-i index_filename]
```

In the following example, a configuration file is removed from a `cfg` clause and then, a file is removed from all of the `cfg` clauses for one release:

```
$ cat INDEX
...
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}=TRUE
cfg "old testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
```

```
        "/var/tmp/config_a"
}
$ manage_index -t -f "/opt/ignite/data/Rel_B.11.11/config" -c "testing" \
> -i /var/tmp/INDEX
$ cat INDEX
...
cfg "testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/var/tmp/config_a"
}=TRUE
cfg "old testing two" {
        description "testing clause"
        "/var/tmp/config_c"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
```

In the previous example, you removed a specific configuration file from the `cfg` clause `"testing"`. The following example removes `config_c` from all B.11.11 clauses:

```
$ manage_index -t -f /var/tmp/config_c -r B.11.11 \
> -i /var/tmp/INDEX
$ cat INDEX
...
cfg "testing two" {
        description "testing clause"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/var/tmp/config_a"
```

```
}=TRUE
cfg "old testing two" {
        description "testing clause"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
```

The `cfg` clause `"testing"` is not modified in this example. Earlier in the section Adding a configuration file to a clause or "release" on page 5, we discussed how manage_index associates a release identifier with a `cfg` clause. The configuration files associated with the `cfg` clause `"testing"` no longer meets any of the conditions that allows `manage_index` to determine a release identifier. Since no release identifier could be determined, the configuration file `/var/tmp/config_c` was not removed from the `cfg` clause.

Important:
A `cfg` clause must have a release keyword in one of the configuration files associated with it.

Removing a script from an **INDEX** file

Earlier you learned how to add a script into the configuration so that it appears on the **Advanced** tab in the Ignite-UX GUI. You use the following form of the command to remove the script:

```
manage_index -t -s script_file_name [-p] [-v] [-i index_filename]
```

The following example adds a script and then removes it:

```
$ manage_index -a -s /var/tmp/script_a -i /var/tmp/INDEX
$ cat INDEX
...
cfg "testing two" {
        description "testing clause"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/var/tmp/config_a"
}=TRUE
cfg "old testing two" {
        description "testing clause"
```

```
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
scripts {
        "/var/tmp/script_a"
}
$ manage_index -t -s /var/tmp/script_a -c "testing" \
> -i /var/tmp/INDEX
$ cat INDEX
cfg "testing two" {
        description "testing clause"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
cfg "testing" {
        description "testing clause"
        "/var/tmp/config_c"
        "/var/tmp/config_a"
}=TRUE
cfg "old testing two" {
        description "testing clause"
        "/opt/ignite/data/Rel_B.11.11/config"
        "/var/tmp/config_a"
}
```

Files (or in this case a script) must exist before they can be removed (or added) with `manage_index`. This applies to configuration files as well as scripts.

```
$ manage_index -a -s /var/tmp/script_a -i /var/tmp/INDEX
ERROR: Cannot read script file: /var/tmp/script_a
ERROR: No read access to file specified by -f.
```

List the names of all configuration files in a **`cfg`** clause

The following command enables you to list all the configuration files named in a `cfg` clause:

```
manage_index -w -c cfg_clause_name [-v] [-i index_filename]
```

For example:

```
$ manage_index -w -c "testing" -i /var/tmp/INDEX
/var/tmp/config_c
/var/tmp/config_a
```

In conjunction with the earlier form of `manage_index` used to list out the names of `cfg` clauses in an `INDEX` file, the preceding command can be used to ensure that all of the configuration files referenced in an `INDEX` file are present and, as far as can be determined, correct.

Display the description of a **`cfg`** clause

This last form of `manage_index` is used to display the description of the `cfg` clause:

```
manage_index -x -c cfg_clause_name [-v] [-i index_filename]
```

For example:

```
$ manage_index -x -c "testing" -i /var/tmp/INDEX
testing clause
```

Using the **`make_bundles`** command

Why do you need to use **`make_bundles`**?

The `make_bundles` command is necessary when using Ignite-UX because commands like `make_config` only work with software that is contained within a bundle. Therefore, if you create a configuration file for a depot and it contains software that is packaged only as an SD product, it is not listed in a configuration file created by `make_config`.

Choosing which form of **`make_bundles`** to use

The `make_bundles` command has the following three forms:

```
/opt/ignite/bin/make_bundles {-b|-B} [-i] [-n name] [-t title] [-c
    category] [-o psf] [-r revision] depot_path

/opt/ignite/bin/make_bundles [-b] [-p|-f] [-i] [-c category] [-o psf]
    depot_path

/opt/ignite/bin/make_bundles {-p|-f|-B} [-i] [-c category] [-o psf]
    [-l file| product/fileset...] depot_path
```

The make_bundles command is primarily controlled by the -B or -b options.[7] The -b option causes make_bundles to operate on all product filesets[8] that are not contained within a bundle.[9] The -B option causes all product filesets in the depot (or those given on the command line) to be operated on regardless of whether they are currently in a bundle or not. The different forms of the make_bundles command, which are covered in this section, create either one or many bundles. Each form has advantages over the other forms depending on what you want to achieve.

You can use the -b option when you want to operate on all of the unbundled product filesets in a depot and the -B if you want to operate on all product filesets within a depot.

You can use the -p and -f options to modify what the -b option does (although you cannot use them with the -B option). The -p option creates one bundle for every product in the depot, making -B unnecessary. When used with the -b option, the command only processes products that are not already contained within a bundle. The -f option has a similar function to -p but it operates at the fileset level; that is, when you use -f it causes a bundle to be created for all filesets in the depot (except when -b is used as well; then only filesets in products not already contained with a bundle are processed).

The form of make_bundles that you use depends upon what you want to do with it. The following sections demonstrate what can be done with three forms of the command.

The make_bundles first form

The first form of make_bundles uses -b or -B and not -p or -f, so you can operate either on all unbundled product filesets or on all product filesets. This form, however, can only create *one* bundle wrapper.

```
/opt/ignite/bin/make_bundles {-b|-B} [-i] [-n name] [-t title] [-c
    category] [-o psf] [-r revision] depot_path
```

With the addition of the -i option, you can set the is_reference SD bundle attribute to TRUE. When set to TRUE, the is_reference causes the bundle wrapper to be installed whenever a product or fileset within a bundle is installed. When set to FALSE, the bundle wrapper is installed only when selected. HP-UX patch bundles always have the is_reference set to TRUE because they can then be installed with either patch_match_target or autoselect_patches set to TRUE. In contrast, if is_reference is set to FALSE, the bundle wrapper for the the HP-UX patch bundle would not be installed with the patches since SD selects patches to be installed, not bundles (leaving the patches unbundled after installation). This allows the bundle wrapper to be automatically installed when patches have been automatically selected for installation by SD. (This can happen if there are software bundles in the same depot as the patches that are installed, and the patches are for filesets in the bundles.)

---

[7] In some forms of make_bundles, the use of -b or -B is optional. However, in the most useful forms of the make_bundles command, you will almost always use one or the other option.
[8] You cannot have a standalone fileset because it must be contained within a product.
[9] If you perform a swlist operation on a depot, the default output shows all bundles in the depot followed by all products not contained within a bundle. The -b option operates only on products not contained within a bundle.

Important:
You should always use the `-i` option when defining bundles that contain patches so that the bundle definition is installed when any patch from the bundle is installed.

With the `-n` option, you can give the bundle a name, which must be 16 characters or fewer. With the `-t` option, you can give a longer descriptive title for the bundle, which must be 256 characters or fewer and can only contain one line. With the `-c` option you can assign a category to the bundle; the default, if not given, is to make the bundle uncategorized. See the section "Categories and other Ignite-UX software attributes" on page 121 for more information on software categories.

With the `-o` option, `make_bundles` writes out a Product Specification File (PSF) instead of modifying the target depot. This option is useful if you need to make customizations to the PSF before applying the changes to the depot. The PSF contains the `swpackage` command required to package the bundle in the depot. The example `swpackage` command contains the path given to PSF "as is". You may need to change the path to the PSF file in the example if you change your current working directory or move the PSF file.

With the `-r` option, you can specify a revision for a bundle. This allows you to create bundle wrappers with revision numbers, enabling you to better manage changes to software.

The examples in this section operate on the following depot:

```
# swlist -s /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#


#
# No Bundle(s) on test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
# Product(s):
#


  PHCO_21187    1.0            cumulative SAM/ObAM patch
  PHCO_23651    1.0            fsck_vxfs(1M) cumulative patch
  PHKL_18543    1.0            PM/VM/UFS/async/scsi/io/DMAPI/JFS/perf patch
  PHKL_20016    1.0            2nd CPU not recognized in G70/H70/I70
  PHKL_22589    1.0            LOFS, select(), IDS/9000 and umount race fix
  PHKL_27980    1.0            VxFS 3.1 cumulative patch: CR_EIEM
  PHKL_28766    1.0            Probe,IDDS,PM,VM,PA-8700,AIO,T600,FS,PDC,CLK
```

Because there are no existing bundles, using the `-b` option or the `-B` options achieves the same result.

The following example bundles all the patches:

```
# make_bundles -b -i -n "PB_July_2004" -t "Site patches for July 2004" \
> -r 1.0 /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
Generating list of unbundled filesets...

=======  07/15/04 00:51:40 EST  BEGIN swpackage SESSION

       * Session started for user "root@test.aus.hp.com".

       * Source:        test:/var/tmp/psf.18564
       * Target:
         test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
       * Software selections:
           *


       * Beginning Selection Phase.
       * Reading the Product Specification File (PSF)
         "/var/tmp/psf.18564".
       * Reading the bundle "PB_July_2004" at line 11.

       * Selection Phase succeeded.


       * Beginning Analysis Phase.
       * Analysis Phase succeeded.


       * Beginning Package Phase.

       * Packaging the bundle
         "PB_July_2004,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
       * Package Phase succeeded.

=======  07/15/04 00:51:41 EST  END swpackage SESSION
```

With `swlist`, you can verify that all of the patches are now contained within the bundle:

```
# swlist -s /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#

#
# Bundle(s):
#

  PB_July_2004          1.0            Site patches for July 2004
```

To continue the example, assume you missed a patch from the bundle that you created and you still need to add it in. In this case, you need to use the `-B` option in a similar command to what was used to create the bundle initially. Therefore, after you copy in the latest `pax` patch `PHCO_30150`, you have the following in the depot:

```
# swlist -s /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#

#
# Bundle(s):
#

  PB_July_2004                  1.0            Site patches for July 2004
#
# Product(s) not contained in a Bundle:
#

  PHCO_30150                    1.0            pax(1) cumulative patch
```

You can then run `make_bundles` again.[10]

```
# make_bundles -B -i -n "PB_July_2004" -t "Site patches for July 2004" \
> -r 1.1 /var/opt/ignite/depots/Rel_B.11.00/patches_11.0

=======  07/15/04 00:57:24 EST  BEGIN swpackage SESSION

       * Session started for user "root@test.aus.hp.com".

       * Source:        test:/var/tmp/psf.18612
       * Target:
         test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
       * Software selections:
           *


       * Beginning Selection Phase.
       * Reading the Product Specification File (PSF)
         "/var/tmp/psf.18612".
       * Reading the bundle "PB_July_2004" at line 11.

       * Selection Phase succeeded.


       * Beginning Analysis Phase.
       * Analysis Phase succeeded.


       * Beginning Package Phase.

       * Packaging the bundle
         "PB_July_2004,r=1.1,a=HP-UX_B.11.00_32/64,v=HP".
       * Package Phase succeeded.

=======  07/15/04 00:57:25 EST  END swpackage SESSION
```

[10] The reason for specifying `-r 1.1` is to identify a collection of patches by a unique bundle specification composed of the bundle name and revision. Revision 1.0 of the bundle `PB_July_2004` does not include `PHCO_30150`; revision 1.1 does include `PHCO_30150`. If the option `-r 1.0` were specified, the original bundle would be replaced.

Now the depot lists two revisions of the bundle:

```
# swlist -s /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#

#
# Bundle(s):
#

  PB_July_2004          1.0            Site patches for July 2004
  PB_July_2004          1.1            Site patches for July 2004
```

By looking at the filesets for revision `1.1` or `PB_July_2004`, you can also verify that the new bundle was created with the correct contents:

```
swlist -l fileset -s /var/opt/ignite/depots/Rel_B.11.00/patches_11.0 \
> PB_July_2004,r=1.1
# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#

# PB_July_2004                              1.1     Site patches for July 2004
# PB_July_2004.PHCO_21187                   1.0     cumulative SAM/ObAM patch
  PB_July_2004.PHCO_21187.INETSVCS-BOOT     1.0     InternetSrvcs.INETSVCS-BOOT
...
# PB_July_2004.PHCO_30150                   1.0     pax(1) cumulative patch
...
```

In the preceding example, note that most of the output is not shown; the excerpt simply demonstrates that the `pax` patch `PHCO_30150` has a member revision `1.1` of the bundle.

If you wanted the pax patch `PHCO_30150` to have its own wrapper bundle, you could have instead used `-b` and changed the bundle name, revision, and so forth and created a new bundle to hold the patch.

The following example demonstrates what happens when you use the `-o` option to create a PSF. It also shows how you can then use the PSF with `swpackage` to implement the changes yourself.

```
# make_bundles -b -i -n "PB_July_2004" -t "Site patches for July 2004" \
> -o ./PB-July_2004.psf -r 1.0 /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
Generating list of unbundled filesets...
# ll ./PB-July_2004.psf
-rw-r--r--   1 root       sys           1138 Jul 15 01:09 ./PB-July_2004.psf
```

The PSF contains all of the information needed to create the bundle and the `swpackage` command needed to create the bundle.[11]

```
# cat PB-July_2004.psf
#####################################################################
# swpackage product specification file generated by make_bundles
# on Thu Jul 15 01:09:42 EST 2004.  For depot:
/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#
# To run swpackage to apply the bundle definitions to the
# depot, use the command:
#
#    swpackage -s ./PB-July_2004.psf -xlayout_version=1.0 -
xreinstall_files=true -d /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#
#####################################################################
bundle
tag PB_July_2004
title Site patches for July 2004
    os_name HP-UX
    is_reference  true
    revision 1.0
    architecture HP-UX_B.11.00_32/64
    vendor_tag "HP"
    contents PHCO_21187,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHCO_23651,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_18543,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_20016,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
```

[11] Getting the `make_bundles` command to produce a PSF also means that the file can be kept in a source management system so changes can be tracked and monitored. Because the PSF can be checked out, the same configuration can be validated and recreated at will.

```
    contents PHKL_22589,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_27980,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_28766,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
end
```

You can then package the bundle manually:

```
# swpackage -s ./PB-July_2004.psf -xlayout_version=1.0 \
> -xreinstall_files=true \
> -d /var/opt/ignite/depots/Rel_B.11.00/patches_11.0

=======  07/15/04 02:34:41 EST  BEGIN swpackage SESSION

       * Session started for user "root@test.aus.hp.com".

       * Source:        test:./PB-July_2004.psf
       * Target:
         test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
       * Software selections:
            *


       * Beginning Selection Phase.
       * Reading the Product Specification File (PSF)
         "./PB-July_2004.psf".
       * Reading the bundle "PB_July_2004" at line 11.

       * Selection Phase succeeded.


       * Beginning Analysis Phase.
       * Analysis Phase succeeded.


       * Beginning Package Phase.

       * Packaging the bundle
         "PB_July_2004,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
       * Package Phase succeeded.
```

```
=======  07/15/04 02:34:42 EST  END swpackage SESSION
```

You can now swcopy the patch PHCO_30150 into the depot (as performed earlier), make changes to the PSF, run the swpackage command again, and repackage the bundle to include the new patch, as follows:

```
# cat PB-July_2004.psf
#####################################################################
# swpackage product specification file generated by make_bundles
# on Thu Jul 15 01:09:42 EST 2004.  For depot:
/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#
# To run swpackage to apply the bundle definitions to the
# depot, use the command:
#
#    swpackage -s ./PB-July_2004.psf -xlayout_version=1.0 -
xreinstall_files=true -d /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#
#####################################################################
bundle
tag PB_July_2004
title Site patches for July 2004
    os_name HP-UX
    is_reference  true
    revision 1.0
    architecture HP-UX_B.11.00_32/64
    vendor_tag "HP"
    contents PHCO_21187,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHCO_23651,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_18543,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_20016,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_22589,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_27980,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHKL_28766,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
    contents PHCO_30150,r=1.0,a=HP-UX_B.11.00_32/64,v=HP,fr=1.0
end
# swpackage -s ./PB-July_2004.psf -xlayout_version=1.0 \
> -xreinstall_files=true \
> -d /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
```

```
=======  07/15/04 02:37:05 EST  BEGIN swpackage SESSION

       * Session started for user "root@test.aus.hp.com".

       * Source:        test:./PB-July_2004.psf
       * Target:
         test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
       * Software selections:
           *


       * Beginning Selection Phase.
       * Reading the Product Specification File (PSF)
         "./PB-July_2004.psf".
       * Reading the bundle "PB_July_2004" at line 11.

       * Selection Phase succeeded.


       * Beginning Analysis Phase.
NOTE:    Repackaging the bundle "PB_July_2004".  (The same bundle
         version already exists in the target depot.)
       * Analysis Phase succeeded.


       * Beginning Package Phase.

       * Packaging the bundle
         "PB_July_2004,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
       * Package Phase succeeded.

=======  07/15/04 02:37:06 EST  END swpackage SESSION
```

Running `make_bundles` with the `-B` option to repackage the bundle has the same effect:

```
# make_bundles -B -i -n "PB_July_2004" -t "Site patches for July 2004" \
> -r 1.0 /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
```

The make_bundles second form

The second form of make_bundles uses the -b option. The -b option causes make_bundles to operate on all products and filesets that are not contained within a bundle.

```
/opt/ignite/bin/make_bundles [-b] [-p|-f] [-i] [-c category] [-o psf]
            depot_path
```

The -p and -f options specify that multiple bundles are to be created.

- If -p is specified and -b is not, a bundle is created for each product in the depot.
- If both -p and -b are specified, a bundle is created for every product in the depot which does not already belong to a bundle.
- If -f is specified and -b is not, a bundle is created for each fileset in the depot.
- If both -f and -b are specified, a bundle is created for each fileset in the depot which does not already belong to a bundle.

The -i, -c, and -o options are discussed in the "The make_bundles first form" section.

The following examples show what happens when you use the -p and -f options with the make_bundles command. These examples do not have a depot with any bundles, so the -b option has no effect.

The first example uses the -p option to show bundles being produced at the product level:

```
# make_bundles -b -p -i /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
Generating list of unbundled filesets...
Generating swpackage PSF...

=======  07/15/04 02:52:16 EST  BEGIN swpackage SESSION

        * Session started for user "root@test.aus.hp.com".

        * Source:        test:/var/tmp/psf.18882
        * Target:
          test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
        * Software selections:
              *



        * Beginning Selection Phase.
        * Reading the Product Specification File (PSF)
          "/var/tmp/psf.18882".
        * Reading the bundle "b_PHCO_21187" at line 11.
        * Reading the bundle "b_PHCO_23651" at line 20.
```

```
        * Reading the bundle "b_PHKL_18543" at line 29.
        * Reading the bundle "b_PHKL_20016" at line 38.
        * Reading the bundle "b_PHKL_22589" at line 47.
        * Reading the bundle "b_PHKL_27980" at line 56.
        * Reading the bundle "b_PHKL_28766" at line 65.

        * Selection Phase succeeded.


        * Beginning Analysis Phase.
        * Analysis Phase succeeded.


        * Beginning Package Phase.

        * Packaging the bundle
          "b_PHCO_21187,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
        * Packaging the bundle
          "b_PHCO_23651,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
        * Packaging the bundle
          "b_PHKL_18543,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
        * Packaging the bundle
          "b_PHKL_20016,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
        * Packaging the bundle
          "b_PHKL_22589,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
        * Packaging the bundle
          "b_PHKL_27980,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
        * Packaging the bundle
          "b_PHKL_28766,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
        * Package Phase succeeded.

=======  07/15/04 02:52:18 EST  END swpackage SESSION
```

The preceding example covers packaging of all the patches in the depot.

Next, the same command is used but with the `-f` option instead of `-p`. All of the bundle names are produced at the fileset level:

```
# make_bundles -b -f -i /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
Generating list of unbundled filesets...
Generating swpackage PSF...
```

```
=======  07/15/04 02:56:18 EST  BEGIN swpackage SESSION

       * Session started for user "root@test.aus.hp.com".

       * Source:        test:/var/tmp/psf.18983
       * Target:
         test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
       * Software selections:
           *


       * Beginning Selection Phase.
       * Reading the Product Specification File (PSF)
         "/var/tmp/psf.18983".
       * Reading the bundle "b_PHCO_21187_INE" at line 11.
       * Reading the bundle "b_PHCO_21187_OBA" at line 20.
       * Reading the bundle "b_PHCO_21187_SAM" at line 29.
...
       * Reading the bundle "b_PHKL_28766_COR" at line 254.
       * Reading the bundle "b1_PHKL_28766_CO" at line 263.
       * Reading the bundle "b2_PHKL_28766_CO" at line 272.

       * Selection Phase succeeded.


       * Beginning Analysis Phase.
       * Analysis Phase succeeded.


       * Beginning Package Phase.

       * Packaging the bundle
         "b1_PHCO_21187_SA,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
...
       * Packaging the bundle
         "b_PHKL_28766_C_I,r=1.0,a=HP-UX_B.11.00_32/64,v=HP".
       * Package Phase succeeded.

=======  07/15/04 02:56:21 EST  END swpackage SESSION
```

Note:
The preceding example merely demonstrates how the -f option works; patches need to be installed fully at the product level not at the fileset level.

The make_bundles third form

The third form of make_bundles uses -p, -f, or -B (all optional) with either a list of products and filesets or a list contained in a file given with the -l option to process. This form of make_bundles gives you improved control over exactly what is packaged into a bundle regardless of what fileset in which it may already be bundled.

```
/opt/ignite/bin/make_bundles {-p|-f|-B } [-i] [-c category] [-o psf]
 [-l file| product/fileset...] depot_path
```

Note:
With this form of the command you cannot give the bundles you are creating a name. The most useful form of this command involves giving the -o option to create a PSF that can be used for subsequent packaging.

In this example, you create some of the same bundles that were created in previous examples. However, this time you use an explicit list of products to include (or patches in this case):

```
# make_bundles -B -i -o ./PB_July_2004_1.0.psf -r 1.0 PHCO_21187 PHCO_23651 \
> PHKL_18543 PHKL_20016 PHKL_22589 PHKL_27980 PHKL_28766 \
> /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
Generating swpackage PSF...
```

Next, you create another PSF but this time with the extra pax patch PHCO_30150:

```
# make_bundles -B -i -o ./PB_July_2004_1.1.psf -r 1.1 PHCO_21187 PHCO_23651 \
> PHKL_18543 PHKL_20016 PHKL_22589 PHKL_27980 PHKL_28766 PHCO_30150 \
> /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
Generating swpackage PSF...
```

You can now apply the commands stored in these two PSF files against the depot. When you do, you end up with the same results as with the make_bundles first form:

```
# swlist -s /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
```

```
# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.00/patches_11.0
#


#
# Bundle(s):
#

  patches_110   1.1            /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
  patches_110   1.0            /var/opt/ignite/depots/Rel_B.11.00/patches_11.0
```

With the third form, you cannot specify an explicit name. You must edit the PSF to change the name of the bundle.

Tip:
If you use change control it is better to generate a PSF and place it under change control than to use `make_bundles` to generate a temporary PSF and apply the changes to the depot. By placing the PSF in your change process, you know exactly how a bundle wrapper in a depot was created and what was in it.

Changes that can impact **make_bundles**

In general, re-executing a `make_bundles` command exactly as it was issued before (same options and arguments) will replace the bundle(s) created during the initial execution.

There is one exception to this behavior: if the initial `make_bundles` command was issued on Ignite-UX B.3.7.x or earlier, and the second `make_bundles` command was issued on Ignite-UX B.3.8.x or later, the earlier bundles are not replaced. Instead, new bundles are created with the same contents and the additional bundle attribute `vendor_tag`, determined from the `vendor_tag` of the products and filesets that are being bundled.

The following example shows a depot containing two such bundles: one created with Ignite-UX B.3.7.x or earlier; and one created with Ignite-UX B.3.8.x or later. Consider the following depot:

```
# swlist -d @/depot
# Initializing...
# Contacting target "akhnaten"...
#
# Target:  akhnaten:/depot
#
```

```
#
# Bundle(s):
#

  Patches       A.1.0          Extended Patch Bundle
  Patches       A.1.0          Extended Patch Bundle
```

This appears to be incorrect because the same bundle appears twice, but it is not. Consider the following:

- A bundle can be specified as follows:

  bundle[,version]

- The version component of the bundles' software specification can be comprised of the following:

  [,r <op> revision][,a <op> arch][,v <op> vendor]

  [,c <op> category][,l=location]

- The <op> (relational operator) can be one of the following:

  =, ==, >=, <=, <, >, or !=

These perform individual comparisons on dot-separated fields. For example, r>=B.10.00 chooses all revisions greater than or equal to B.10.00. The system compares each dot-separated field to find matches. Therefore, a software specification that matches both bundles is Patches,r=A.1.0. Additionally, the specification could include the a (architecture), v (vendor), c (category), l (installation location), fr (fileset revision when referring to filesets only), or fa (fileset architecture when referring to filesets only).

When make_bundles (as delivered in version B.3.7.x or earlier) was executed to create the first bundle, the output looked like this:

```
# make_bundles -B -r A.1.0 -n Patches -t "Extended Patch Bundle" /depot
...
   * Packaging the bundle "Patches,r=A.1.0,a=HP-UX_B.11.00_32/64".
...
```

However, when make_bundles (Ignite-UX version B.3.8.x or later) was executed to create the second bundle, the output looked like this (note the addition of v=HP):

```
# make_bundles -B -r A.1.0 -n Patches -t "Extended Patch Bundle" /depot
...
   * Packaging the bundle
   "Patches,r=A.1.0,a=HP-UX_B.11.00_32/64,v=HP".
...
```

Both bundles show the bundle name and the architecture of the bundle, but only the newer `make_bundles` command shows the vendor as part of the bundle software specification. You can verify this by looking at the depot again:

```
# swlist -d -l bundle -a name -a revision -a vendor_tag @/depot
# Initializing...
# Contacting target "akhnaten"...
#
# Target:  akhnaten:/depot
#

  Patches       A.1.0
  Patches       A.1.0          HP
```

You can see that the depot definitely contains distinct bundles: one without the `vendor_tag` attribute set and the second with it to `HP` as shown in the preceding example.

To remove both bundles would execute the following command twice, once for each bundle:

```
swmodify -u Patches,r=A.1.0 @/depot
```

Other variations of the `swmodify` command do not operate as expected so HP recommends that you use the command in this manner. Optionally, you could use `swlist` to verify if the older bundle was removed.

You do have other choices in these circumstances, as previously discussed. If you have kept the product specification file (PSF) file created by the older `make_bundles` command, you could update it to add the new products and filesets you want to include in the bundle.

Alternatively, you could create a PSF file with the *new* `make_bundles` command and remove the `vendor_tag` information from the PSF before running `swpackage`. This enables you to update the bundle information created by the older `make_bundles` command.

Using the **`instl_dbg`** command

Introduction

The `instl_dbg` command is often overlooked but offers valuable functionality in testing the effect of a change in a configuration file. This section introduces some of the useful functionality of `instl_dbg` and how it can be combined with the `itool` command to rapidly prototype and test configuration changes.

Requirements

The `instl_dbg` command requires some information that can only be created during an installation session either by adding a new client to an Ignite-UX server, by running `make_net_recovery`, or by running `make_tape_recovery` with `the -s` option. The files that are most needed are `hw.info`, `host.info`, and `config.sys`.

```
/opt/ignite/bin/instl_dbg -D client_directory [-f file] [-v] [-a {l|r}]
[-cdls|-A] [-V var[=value]] [-S swsel[=TRUE|=FALSE]]
[-U use_model[=TRUE|=FALSE]] [-?]
```

The one command-line option required by `instl_dbg` is the `-D` option. This would usually point to a per-client directory on an Ignite-UX server. It must contain all of the information that `instl_dbg` needs to run. If you add a new client to an Ignite-UX server, the information you need is created automatically. However, this does not help when you have a new client that has never had an operating system installed.

Using the **`itool`** command

The `itool` command provides the Wizard and Advanced interfaces during recovery and installation. It is not well known that the `itool` command can be run from the command line on an Ignite-UX server to manipulate the configuration for a client. The `itool` command accepts the following options[12]:

```
# itool -?
itool: illegal option -- ?
usage itool [ arguments ]
          -d  <directory path>
          -i  <client name>
          -m 'demo', 'push', 'pull', 'quick', or 'wiz'
          -w show welcome screen in push mode
          (The default is -m pull)
```

The only method you really need to worry about when using `itool` is as follows:

```
# itool -d <client dir> -m pull
```

This runs the `itool` user interface as though it were running on the client system.[13] You must run `itool` as `root`.

Important:
Do not change time-related information in the Ignite-UX GUI unless the system you are on is considered non-production; altering the time information changes the system clock.

[12] HP does not recommend the use of `itool` outside of its normal use (running on a client system during installation or recovery or via the Ignite user interface).*`ignite`*`(1)`). This information is presented here so `itool` can be run in conjunction with `instl_dbg` for rapid prototyping of configurations.

[13] When an installation is controlled from the server, `itool` is actually run on the server. However, it is run with the `-m` push option since configuration changes are pushed to the client system.

Configuration changes are saved into the configuration file in the per-client directory.

When exiting `itool`, there no need for concern with respect to the list of options it presents; no action should be taken for any of them. The different options set a different return code from `itool`. The return code is how `itool` communicates the next action to the program that calls it when it is running during a normal installation or recovery.

Combining **instl_dbg** and **itool**

If a configuration file (a file named `config`) exists in the client directory, its configuration information is used (for example, which `cfg` clause) instead of the default. This is the reason the `itool` program was discussed in the previous section.

When you use `itool` to change configurations and then run `instl_dbg` to see what is happening in the configurations, it makes it possible to test configurations and changes without having to boot a system and attempt to install the system.

Running `instl_dbg` in this way enables you to see the configuration the way Ignite-UX parses it on that client.

Running **instl_dbg**

You can use `instl_dbg` to test some aspects of your configuration; for instance, with the `-c` option you can perform system configuration sanity checks. In the following example, it is assumed that you have changed working directories to a per-client directory on an Ignite-UX server (`/var/opt/ignite/clients/<MAC>`).

```
# instl_dbg -D . -c

======= System Configuration Checks ======
WARNING: The disk at: 0/8/0/0.2.0 (SEAGATE_ST39173WC) appears to contain a
file
         system and boot area.  Continuing the installation will destroy any
         existing data on this disk.
```

The `-c` option allows you to see configuration check messages that would normally only be seen during an installation or recovery session. This allows you to preview what you will see during an installation or recovery session.

With the `-d` option, you can get `instl_dbg` to print out the expected size of the volumes it creates and what the impacts are for each file system. The expected output of percent (%) used is also shown with the file system type. If the disk does not contain enough space, the `amount shy (KB)` column indicates the additional space needed.

```
# instl_dbg -D . -d
======= Disk and Volume allocation ======
  Mount          Size     Usage     Disk /       impact      amount
Directory        (Mb)               Group      (MB /  %)    shy (KB)
```

```
-------------------------------------------------------------------------
/stand            300      HFS       vg00        21   7%         0
(swap_dump)       1280     SWAP_DUMP vg00         0   0%         0
/                 200      VxFS      vg00        92  46%         0
/tmp              200      VxFS      vg00         0   0%         0
/home             20       VxFS      vg00         0   0%         0
/opt              976      VxFS      vg00       723  74%         0
/usr              1296     VxFS      vg00      1035  79%         0
/var              1780     VxFS      vg00       229  12%         0
-------------------------------------------------------------------------
Size unallocated from group: vg00: 2624Mb
       * Expected physical allocation of volumes on disks:
       *     0/8/0/0.2.0 (/dev/dsk/c0t2d0) (vg00)
       *             3997 -   311197 KB  /stand
       *           311197 -  1621917 KB  primary
       *          1621917 -  1826717 KB  /
       *          1826717 -  2031517 KB  /tmp
       *          2031517 -  2051997 KB  /home
       *          2051997 -  3051421 KB  /opt
       *          3051421 -  4378525 KB  /usr
       *          4378525 -  6201245 KB  /var
       *          2686976 KB unallocated
```

The `-l` option to `instl_dbg` performs some lint type checks on the configuration. Variables that are not assigned with the `init` keyword appear as follows:

```
# instl_dbg -D . -l
...
******* Variables assigned without init keyword *******
       * _my_variable
```

Note:
Every variable used in an Ignite-UX configuration file should be initialized using the `init` keyword. If you do not want the variable to be changed, use the `visible_if` keyword to prevent it from being displayed by the Ignite-UX GUI. (See *instl_adm*(1M) for more information about the `init` and `visible_if` keywords.)

```
# instl_dbg -D . -l
```

```
======= Lint-type Checks =======

******* Variables assigned without init keyword *******
All variables assigned with the init keyword.
```

With the `-s` option, the value of all variables, `sw_sel` objects, and usage models are shown.

The information is quite detailed. From the information for `sw_sel` objects, you can tell if `sw_sel` has been selected, if it has been selected because of a dependency, or if it has been unselected because of an exrequisite.

The variable information provides the name of the variable, the value that it holds, the type of variable, and whether the variable is visible (based on the setting of visible_if). If the variable has an "effects" relationship, the name of the variable it has a relationship with is shown, and if the variable has a list of potential values, the list of values is shown.

For use models, the name is shown, and whether it is selectable, selected, or visible.

```
# instl_dbg -D . -s

======= Software Selection Attributes: ======
sw_sel      name: "100BaseT-01"
     description: "HP-PB 100BaseT;Supptd HW=A3495A;SW=J2759BA"
        selected: 0
    dep_selected: 0
   ex_unselected: 0
      is_defined: 1

sw_sel      name: "100BaseT-00"
     description: ""
        selected: 0
    dep_selected: 0
   ex_unselected: 0
      is_defined: 1
...
sw_sel      name: "perl"
     description: "Perl Programming Language"
        selected: 0
    dep_selected: 1
   ex_unselected: 0
      is_defined: 1

======= Variable Attributes: ======
```

```
Variable  name: "_hp_boot_dev_path"
         value: "0/8/0/0.2.0"
          type: string
       visible: 0

Variable  name: "_hp_default_cur_lan_dev"
         value: "lan5"
          type: string
       visible: 0
...
Variable  name: "_hp_disk_layout"
         value: "Logical Volume Manager (LVM) with VxFS"
          type: string
       visible: 0
   val_list[0]: "Whole disk (not LVM) with HFS"
   val_list[1]: "Logical Volume Manager (LVM) with HFS"
   val_list[2]: "Logical Volume Manager (LVM) with VxFS"
   val_list[3]: "VERITAS Volume Manager (VxVM) with VxFS"
   Effects var: _hp_pri_swap
   Effects var: _hp_group_name

Variable  name: "_hp_root_disk"
         value: "0/8/0/0.2.0"
          type: string
       visible: 0
   Effects var: _hp_pri_swap
   Effects var: _hp_min_swap
   Effects var: _hp_disk_layout
   Effects var: _hp_root_grp_disks
...
======= Use-Model Attributes: ======
Use model name: "Create /export volume"
      selected: 0
    selectable: 1
       visible: 1

Use model name: "Create separate volumes (/usr, /var, ...)"
      selected: 1
    selectable: 1
       visible: 1
```

Using the `-f` option, the `instl_dbg` command also provides a completely parsed version of the configuration. This output builds on top of the configuration files parsed on the system. In this case, the configuration was modified using the Ignite-UX GUI before running `instl_dbg`:

```
# instl_dbg -D . -f ./parsed.output
# cat parsed.output
cfg "HP-UX B.11.11 Default"=TRUE
server="10.0.0.60"
is_net_info_temporary=FALSE
init _hp_keyboard="Not_Applicable"
system_name="spectre"
netmask["lan0"]="255.255.255.0"
ip_addr[]="10.0.0.162"
netmask[]="255.255.255.0"
route_gateway[0]="10.0.0.1"
route_destination[0]="default"
_hp_cfg_detail_level="ibnpvcdsrlLtfh"
#
# Variable assignments
#
init _hp_boot_dev_path="0/8/0/0.2.0"
init _hp_default_cur_lan_dev="lan5"
init _hp_primary_path="0/8/0/0.2.0"
init _hp_current_client_release="B.11.11"
init _hp_HFS_blksize=65536
init _hp_HFS_fragsize=8192
init _hp_VxFS_blksize=8192
init _hp_FS_stripe_size=64K
init _hp_disk_layout="Logical Volume Manager (LVM) with VxFS"
...
init _hp_os_bitness="64"
init _hp_patch_save_files="YES"
init _hp_custom_sys="Current System Parameters"
init _hp_locale="SET_NULL_LOCALE"
init "Create /export volume"=FALSE
init "Create separate volumes (/usr, /var, ...)"=TRUE
sd_command_line=" -x os_release=B.11.11 -x os_name=HP-UX:64 "
#
```

```
# Software Selections
#
init sw_sel "100BaseT-01"=FALSE
init sw_sel "ATM-00"=FALSE
init sw_sel "ATM-01"=FALSE
init sw_sel "B5725AA"=TRUE
...
init sw_sel "b_PHSS_28764"=FALSE
init sw_sel "b_PartitionManag"=FALSE
# sw_sel "perl" will be loaded due to other software.
(cfg "HP-UX B.11.11 Default") {
...
        sw_source "core" {
                description="HP-UX coreSoftware"
                source_type="NET"
                source_format=SD
                sd_server="10.0.0.60"
                sd_depot_dir="/var/opt/ignite/depots/Rel_B.11.11/core"
                sd_command_line=" -xpatch_filter=*.*  -xpatch_save_files=true
"
                sd_use_ui=FALSE
                load_order=0
        }
        sw_source "Kernel Configuration" {
                source_format=CMD
                load_order=11
        }
        sw_source "site commands" {
                source_format=CMD
                load_order=10
        }
}
RUN_UI=TRUE
RUN_SD=TRUE
SD_USE_UI=FALSE
CONTROL_FROM_SERVER=FALSE
HALT_WHEN_DONE=FALSE
USE_EXPERT_UI=TRUE
CLEAN_ALL_DISKS=FALSE
HIDE_BOOT_DISK=FALSE
```

```
ERROR_IF_BAD_SW=FALSE
RECOVERY_MODE=FALSE
DISABLE_DHCP=TRUE
ALLOW_DISK_REMAP=FALSE
release="B.11.11"
#
# System/Networking Parameters
#
_hp_custom_sys+={"Current System Parameters"}
init _hp_custom_sys="Current System Parameters"
_hp_custom_sys visible_if TRUE
_hp_custom_sys help_text "System/Networking Info"
(_hp_custom_sys=="Current System Parameters") {
        final system_name="testa"
        final ip_addr["lan5"]="10.0.0.162"
        final netmask["lan5"]="255.255.255.0"
        final ip_addr[]="10.0.0.47"
        final netmask[]="255.255.255.0"
        final route_gateway[0]="10.0.0.1"
        final route_destination[0]="default"
        TIMEZONE="EST-10EDT"
        ROOT_PASSWORD="/aIc61gXgbz2A"
        is_net_info_temporary=FALSE
        run_setparms=FALSE
} # end "Current System Parameters"

#
# Disk and File systems
#
_hp_disk_layout+={"Modified LVM Layout"}
init _hp_disk_layout="Modified LVM Layout"
(_hp_disk_layout=="Modified LVM Layout") {
        # Disk/File system Layout:
        volume_group "vg00" {
                usage=LVM
                physical_volume disk[_hp_root_disk] {
                } # end pv_options
                max_physical_extents=2500
                logical_volume {
                        mount_point="/stand"
```

```
                        disk[_hp_root_disk]
                        usage=HFS
                        size=307200K
                        FILE_LENGTH=LONG
                        FRAGSIZE=8192
                        BLKSIZE=65536
                        bad_block_relocate=FALSE
                        contiguous_allocation=TRUE
                } # end logical_volume
...
                logical_volume {
                        mount_point="/var"
                        usage=VxFS
                        size=73728K | remaining | 1822720K
                        BLKSIZE=8192
                        largefiles=TRUE
                } # end logical_volume
        } # end volume_group "vg00"
} # end "Modified LVM Layout"
```

Other **instl_dbg** options

The `-vvv` option (very verbose) lists all the selections and values of all variables. You can select use models (for example, "Logical Volume Manager (LVM) with VxFS") and set values for variables to test the impact that a change has on the configuration.

Refer to *instl_dbg*(1M) for more information on other options available.

The **hw.info** and **host.info** files

Both the `hw.info` and `host.info` files can be created in either of two ways:

- When you add a new client to the Ignite-UX server for recovery, the `add_new_client` script creates these two files (indirectly).
- When a new client boots from the Ignite-UX server and creates a directory for itself, information about the client hardware is written to the `hw.info` file.

Creating both files

You can easily create both `host.info` and `hw.info` files on a system by using the same method as the `add_new_client` script. The following example, run on the client system as `root`, places the `host.info` and `hw.info` files in the current working directory:

```
# export INST_CLIENT_DIR=./
# export PATH=$PATH:/opt/ignite/lbin
```

```
# rescan_hw_host
       * Scanning system for IO devices...
       * Querying disk device: 0/0/1/1.15.0 ...
       * Querying disk device: 0/0/2/1.15.0 ...
# ll
total 4
-rw-r--r--   1 root       sys            214 Apr 27 15:31 host.info
-rw-r--r--   1 root       sys            688 Apr 27 15:31 hw.info
# cat host.info
MEMORY=1310720K
HARDWARE_MODEL="9000/800"
MODEL="9000/800/A400-6X"
can_run_32bit=TRUE
can_run_64bit=TRUE
is_numa=FALSE
is_ia64=FALSE
is_hppa=TRUE
_hp_boot_dev_path="0/0/1/1.15.0"
_hp_boot_dev_path visible_if FALSE
# cat hw.info
cdrom: 0/0/1/0.1.0 2 sdisk 188 31 1000 HP_DVD-ROM_304 0 /dev/rdsk/c0t1d0
/dev/dsk/c0t1d0 -1 -1 0 1 0
disk: 0/0/1/1.15.0 0 sdisk 188 31 1f000 SEAGATE_ST336704LC 35566480
/dev/rdsk/c1t15d0 /dev/dsk/c1t15d0 -1 -1 5 1 10
disk: 0/0/2/1.15.0 1 sdisk 188 31 3f000 SEAGATE_ST336704LC 35566480
/dev/rdsk/c3t15d0 /dev/dsk/c3t15d0 -1 -1 4 1 10
lan: 0/0/0/0 0 lan0 btlan 000F201D7D1F HP_PCI_10/100Base-TX_core0
ext_bus: 0/0/1/0 0 c720 n/a SCSI_C896_Ultra_Wide_Single-Ended
ext_bus: 0/0/1/1 1 c720 n/a SCSI_C896_Ultra_Wide_Single-Ended
ext_bus: 0/0/2/0 2 c720 n/a SCSI_C87x_Fast_Wide_Single-Ended
ext_bus: 0/0/2/1 3 c720 n/a SCSI_C87x_Ultra_Wide_Single-Ended
processor: 160 0 processor n/a Processor
```

Note:
The lines are not folded in the actual `hw.info` file as they appear in the preceding example.

You can test configuration changes with `instl_dbg` by copying an existing per-client directory to a new directory. However, it is not a good idea to copy the directory to another directory under `/var/opt/ignite/clients` in case the `<MAC>` directory you choose to copy it to conflicts with a potential future client; instead, copy it somewhere else.

You can either copy the `hw.info` and `host.info` files from the client system or freely change the existing `hw.info` and `host.info` files for testing. For example, the disk size is the field after the description, but before the raw device file name (both disk devices are set to 35566480 in the preceding example).

If you had to test a configuration with 72 GB disks, you could manually change those sizes. While you might consider doing this for testing purposes, you should *never* change the `hw.info` or `host.info` files in a per-client directory on the Ignite-UX server that is used by a "real" client system, as doing so may cause many potentially fatal problems.

Miscellaneous configuration tips

- In Ignite-UX configuration files, `TRUE` and `FALSE` should be stated only in upper- or lower-case and never in mixed case. Ignite-UX does not understand `TRUE` or `FALSE` when they are in mixed case. This applies to other keywords as well.
- Be careful when deciding to use the = or += operators. The += operator enables you to add to something, while the = operator removes any current value for a variable and replaces it with the new value being assigned. See the "`_hp_disk_layout`" section on page 86 for information about how = and += can be used to replace existing disk layouts or add more layout choices. The `error` keyword is a good example of the need to use += at times. If you use =, you remove any previous error messages that may have been encountered. If you use +=, all of the error messages, both previous and new, can be shown to the user.
- When you are using the Ignite-UX GUI to add new logical volumes, you might want to consider choosing the logical volume that is most similar to what you want to create and use that as a starting point. Change the attributes and instead of selecting **Modify**, then select **Add** to create your new logical volume.
- When setting a network configuration, review *`instl_adm`*`(4)` to verify whether the modifier `final` (placed before a keyword, such as `dns_domain`) is required to retain the configuration when the system is installed. If it is required and you do not provide this modifier, the configuration is only kept for the duration of the installation and is discarded at the final system reboot.
- If you do not want to preserve instance numbers when cloning a system, you should remove all of the `hw_instance_num` lines from the `system_cfg` file. Unlike network recovery archives, you cannot remove `hw_instance_num` lines from a recovery tape after creation. Instead, if you use a recovery tape for cloning and do not want to preserve instance numbers, you must preview using `make_tape_recovery` with the `-p` option, edit the associated configuration file (`/var/opt/ignite/recovery/latest/system_cfg`), and then resume creation of the recovery tape using `make_tape_recovery` with the `-r` option. For a network recovery archive, you would find the `system_cfg` file in the following location:

  `/var/opt/ignite/clients/<MAC>/recovery/<DATE-TIME>/system_cfg`

Analyzing the HP-UX default B.11.11 **`cfg`** clause

The following is from a default `/var/opt/ignite/INDEX` showing the configuration files referenced by the HP-UX B.11.11 `cfg` clause:

```
cfg "HP-UX B.11.11 Default" {
        description "This selection supplies the default system configuration
that HP supplies for the B.11.11 release."
        "/opt/ignite/data/Rel_B.11.11/config"
        "/opt/ignite/data/Rel_B.11.11/hw_patches_cfg"
        "/var/opt/ignite/config.local"
}
```

This section describes these configuration files and explains what is happening and why certain things have been done in certain ways.

The release-specific configuration file

The `config` file defines all of the file system and volume management configurations for the release (and a few other things).

```
#############################################################################
# config:
#
# (c) Copyright Hewlett-Packard Company 1994
#
# This file contains the default system definitions used during a
# cold-install of HP-UX.  This file should not be modified.  If
# modifications are needed, the file "config.local" should be modified
# instead.
# See the instl_adm(4) man page for details.
#
# Changes to this file will not be preserved if the fileset is re-installed.
#
# If you are editing this file via the advanced options during a
# cold-install, then notice that the "config.local" contents are appended
# to the end of this file.
#
# @(#) config $Revision: 10.82 $
```

The `release` keyword is probably the single-most important keyword in an Ignite-UX configuration file. This keyword *must* be set or else any installation of HP-UX is likely to fail or not work properly. Ignite-UX has no idea what version of HP-UX it is installing unless it is set.

As of Ignite-UX C.6.0.*x*, the `release` keyword is checked against the value of the `_hp_ikernel_os_release` variable (the client creates the value of this variable in the

host.info file). If the HP-UX release in the release keyword is not supported by the kernel version given in _hp_ikernel_os_release the installation is not allowed to proceed and the following error appears:

```
ERROR:    The version of HP-UX you have chosen to install on the system
          (B.11.23) is not supported by the version of the Ignite-UX install
          kernel that the system booted (B.11.11).  You will need to reboot the
          target system from an install kernel matching the desired release
from
          the menu at the console. If using the bootsys command, use the '-R'
          option to specify the install kernel version.
```

Table 2 shows the different HP-UX releases, beginning with Ignite-UX C.6.0.*x*, which can be installed with the different versions of an installation kernel[14].

Table 2

HP-UX Release	PA-RISC	Itanium®-based
B.11.11	B.11.00 / B.11.11	N/A
B.11.23	B.11.23	B.11.22 / B.11.23

Therefore, if you boot the B.11.11 installation kernel you can install HP-UX B.11.00 and B.11.11 but you cannot install B.11.23. The reverse applies if you boot the B.11.23 PA-RISC installation kernel: you cannot install B.11.11 or B.11.00 with it.

When you create custom configuration files and do not plan to include the default configuration files you *must* set this variable.

```
release="B.11.11"

#############################################################################
```

Sometime ago, changes were made to Ignite-UX to increase the block and fragment sizes for VxFS and HFS file systems. This can lead to differences between systems installed in the last few years and much older systems that had smaller default block and fragment sizes (for example, the file system attributes are different so the block and fragment sizes from older systems are probably smaller).

The use of <variable> visible_if false ensures that the variables do not appear in the **Additional** dialog box from the **Basic** tab.

```
init _hp_HFS_blksize = 65536    # HFS block size.
init _hp_HFS_fragsize = 8192    # HFS frag size.
init _hp_VxFS_blksize = 8192    # VxFS block size.
```

[14] This assumes that you have installed the necessary filesets to support those HP-UX releases.

```
init _hp_FS_stripe_size = 64Kb  # File system stripe size.

_hp_HFS_blksize visible_if false
_hp_HFS_fragsize visible_if false
_hp_VxFS_blksize visible_if false
_hp_FS_stripe_size visible_if false
#############################################################################
```

The following example shows the start of a definition for the `_hp_disk_layout` variable, which is set to one of four possible values.

Around the `_hp_disk_layout` is a test for `is_hppa` (is PA-RISC). This only exists in this file since the file is generated from one common source file using `unifdef` (refer to *`unifdef`*`(1)` for more information) for all HP-UX releases. Obviously, this test is not actually needed for HP-UX B.11.11 since it is only supported on PA-RISC systems. The entire release-specific configuration files are based on one original file and they are processed with `unifdef` to produce the release-specific configuration file.

You actually get to see the value of the variable `_hp_disk_layout` when you use the Ignite-UX GUI. The values currently assigned to `_hp_disk_layout` are shown on the **File System:** list on the **Basic** tab. The form used in the next example to initialize a list of possible values allows the Ignite-UX GUI to modify the list later.

Further in the configuration file, when giving `_hp_disk_layout` a final value you only assign its value using `init`. When using the form "`init _hp_disk_layout="…"`" the Ignite-UX GUI is still allowed to modify the value. If you instead define "`_hp_disk_layout="…""` without the `init`, the Ignite-UX GUI cannot change this value. If the Ignite-UX GUI were not able to change the value of `_hp_disk_layout`, no changes made in the Ignite-UX GUI to file system attributes are applied to the system[15].

This is because the Ignite-UX GUI must change the value of `_hp_disk_layout` to "Modified LVM Layout", "Modified VxVM Layout", or "Modified Whole-Disk Layout" when changes have been made. If the Ignite-UX GUI is not allowed to change the variable, you cannot keep any file system customizations. This concept becomes very important when creating custom configuration files.

```
#############################################################################
# Any _hp_disk_layout variable setting besides those below will disable
# the default disk layout described in this file.  This can be
# done if a custom disk layout is defined elsewhere.
#

# For PA architecture only
is_hppa {
```

[15]The itool command Ignite-UX GUI accepts changes to file system attributes and does not caution you, but no file system attribute changes made in the Ignite-UX GUI are applied to the system.

```
    _hp_disk_layout = {
        "Whole disk (not LVM) with HFS",
        "Logical Volume Manager (LVM) with HFS",
        "Logical Volume Manager (LVM) with VxFS",
        "VERITAS Volume Manager (VxVM) with VxFS"
    }
}
```

You can now set the variable _hp_disk_root to the value of _hp_primary_path, assuming that the device that _hp_primary_path points to exists and _hp_root_disk does not already have a value. You can use an Extended Regular Expression (ERE)[16] to see if _hp_root_disk is set to any value[17].

```
#
# Set the default root disk to the current primary path.  The
# _hp_primary_path will be "" if it was set to a non-existent dev.
# also don't reset the _hp_root_disk if it was initialized in
# a different config file and non-null (see 15654FSDdt).  Use
# the ~ operator instead of == to detect if _hp_root_disk is not
# yet set (See FSDdt22058).
#
(_hp_primary_path != "" & !(_hp_root_disk ~ ".*"))
{
    init _hp_root_disk=_hp_primary_path
}
```

The next section of the configuration file presents a critical concept that it is important to understand – the idea of the effects relationship. An effects relationship creates a relationship between two variables so that when one variable changes the other variable is re-evaluated – even if its final value is not based upon that variable.

In the next example, you can see that the value of _hp_pri_swap has been set to the size of the root disk and to the value of _hp_disk_layout (after that primary swap is initialized to a value more useful for swap).

This forces Ignite-UX to reevaluate the value of _hp_pri_swap whenever the size of the root disk changes (a smaller disk might force Ignite-UX to define a smaller swap space) or when the disk layout changes.

```
#
```

[16] For more information regarding ERE, refer to *regexp*(5).
[17] The ERE ".*" means zero or more of any character so it matches a zero length string.

```
# The next two statements are used only to establish an effects
# relationship between the _hp_pri_swap variable and _hp_root_disk &
# _hp_disk_layout variables.  The value is overwritten below.
# This makes the UI change the swap anytime either of these two
# values change.
init _hp_pri_swap = disk[_hp_root_disk].size
init _hp_pri_swap = _hp_disk_layout

# default (recommended) swap size is 2 X memory
# Round up to the nearest 64MB so that we don't get a warning
# about it being adjusted in the UI.  Really this just needs
# to be rounded to be a multiple of the physical-extent-size, but
# that can vary depending on the size of disk (4,8,16,32,64...)
# 64MB should work on the majority of the disks available.
init _hp_pri_swap = (((MEMORY * 2) + 65535KB)/64MB)*64MB
```

Now you have other calculations for swap for different sized disks. The following example limits the size of swap for small disks. If you use `init` when setting both `_hp_pri_swap` and `_hp_sec_swap`, then the configuration only uses methods for setting the variables that allows the Ignite-UX GUI to modify them.

```
# Put an upper bounds of 1024Mb to the default swap size for disks
# less than 5Gb.  And 4Gb swap for other disks
_hp_pri_swap > 1024Mb {
    disk[_hp_root_disk].size < 5120Mb {
        init _hp_pri_swap = 1024Mb
    } else {
        _hp_pri_swap > 4096Mb { init _hp_pri_swap = 4096Mb }
    }
}

# Use a 128Mb as the default minimum amount of swap configured on any
# system.  The real swap space will be reduced down to _hp_min_swap if
# there is not enough file system space.
(_hp_pri_swap < 128Mb)
    { init _hp_pri_swap = 128Mb }

# Initialize the swap range minimum to what the default is
_hp_min_swap = _hp_pri_swap

# If the system is limited on resources, then reduce the minimum so
```

```
# that the OS has a better chance of fitting.  The swap size will
# still be set to the recommended value if there is enough disk space.
(_hp_min_swap > 512Mb & disk[_hp_root_disk].size < 3000Mb)
        { _hp_min_swap = 512Mb }
(_hp_min_swap > 192Mb & disk[_hp_root_disk].size < 1800Mb)
        { _hp_min_swap = 192Mb }
(_hp_min_swap > 96Mb & disk[_hp_root_disk].size < 700Mb)
        { _hp_min_swap = 96Mb }
(_hp_min_swap > 68Mb & disk[_hp_root_disk].size < 600Mb)
        { _hp_min_swap = 68Mb }
```

Now `_hp_sec_swap` has been initialized with a set of values that can be chosen from; however, they are not a fixed set of values. The variable `_hp_sec_swap` can be set using the **Additional** button because of the way the variable was defined (`<variable>={ value1, value2, …}`). The variable `_hp_sec_swap` has its value set from a selection list or by typing it into the corresponding field directly. This can happen because `_hp_sec_swap` was not defined as an "enum"; if it had been defined as an enum only a value from the list of values it contains could be used.

Although, in this configuration, `hp_pri_swap` cannot be modified using the **Additional** button. This value appears directly in the Ignite-UX GUI on the **Basic** tab using the **Root Swap (MB)** button and it allows you to select from a list of possible values. The list of values shown in the selection list for `_hp_pri_swap` comes from the list defined in the configuration. If you create a custom disk configuration file, you should supply a value or a list of values for `_hp_pri_swap` (or somehow otherwise calculate a default value for it).

```
_hp_pri_swap = { 128Mb, 256Mb, 512Mb, 1024Mb,
                 1536Mb, 2048Mb, 3072Mb, 4096Mb,
                 8192Mb, 12288Mb, 16384Mb }

_hp_sec_swap ={ 0Mb, 128Mb, 256Mb, 512Mb, 1024Mb,
                1536Mb, 2048Mb, 3072Mb, 4096Mb,
                8192Mb, 12288Mb, 16384Mb }
init _hp_sec_swap = 0Mb
_hp_sec_swap help_text "Secondary Swap space (KB)"
```

The next example gives `_hp_disk_layout` a value of disk `[_hp_root_disk]`.model. If the value of that expression ever changes, it causes the value of `_hp_disk_layout` to be re-evaluated.

Doing this makes good sense in the context of this configuration file because if you change the root disk, you may need to re-evaluate what the disk configuration is. Tests later on may override some defaults for `_hp_disk_layout` in the following example:

Note:
The models being tested on are not likely ever to run HP-UX B.11.11. This is old data being carried along into later releases.

```
# This next statement is used only to establish an effects relationship
# between the _hp_disk_layout variable and _hp_root_disk.  The value is
# overwritten below.
init _hp_disk_layout = disk[_hp_root_disk].model

#
# Define the defaults based on the system architecture 700 vs 800
#
is_hppa {
    HARDWARE_MODEL ~ "9000/7.*" {
        init _hp_disk_layout = "Logical Volume Manager (LVM) with VxFS"
    } else {
    # For a certain class of S800's, non-LVM takes >30 minutes to boot due
    # to sequential access firmware.  So make the default LVM on them no
    # matter what size of disk it is.
        disk[_hp_root_disk].size >= 600Mb |
        HARDWARE_MODEL ~ "9000/825.*" |
        HARDWARE_MODEL ~ "9000/834.*" |
        HARDWARE_MODEL ~ "9000/835.*" |
        HARDWARE_MODEL ~ "9000/635.*" |
        HARDWARE_MODEL ~ "9000/845.*" |
        HARDWARE_MODEL ~ "9000/645.*" |
        HARDWARE_MODEL ~ "9000/822.*" |
        HARDWARE_MODEL ~ "9000/832.*" |
        HARDWARE_MODEL ~ "9000/842.*" |
        HARDWARE_MODEL ~ "9000/852.*"
        {
            init _hp_disk_layout = "Logical Volume Manager (LVM) with VxFS"
        } else {
            init _hp_disk_layout = "Whole disk (not LVM) with HFS"
        }
    }
    #
    # For disks over 2Gb+300k the default must be LVM.
    # Set the default here, and sanity_checks will ensure it.
```

```
    #
    disk[_hp_root_disk].size > 2097452K {
        init _hp_disk_layout = "Logical Volume Manager (LVM) with VxFS"
    }
} else {
    init _hp_disk_layout = "VERITAS Volume Manager (VxVM) with VxFS"
}
```

The enumeration `_hp_root_grp_striped` in the following example is a perfect example of how to ask a yes/no or Boolean question using the **Additional** button on the **Basic** tab in the Ignite-UX GUI. The first step is to define the variable as an `enum`, then give it two values, an initial value (with the `init` keyword so the user is allowed to change it using the Ignite-UX GUI), and some help text.

Without defining `_hp_root_grp_striped` as an `enum`, the user would be able to select **YES** or **NO** on the **Additional** button or also enter any value. In this case, it does not matter because the configuration only ever tests on the value **YES** (later) to decide on an action, but if you could enter a lower-case "yes" the tests would fail; hence the enumeration type is used.

```
#
# Boolean describing whether or not the root disk group
# will be striped
#
enum _hp_root_grp_striped
_hp_root_grp_striped = { "YES", "NO" }
init _hp_root_grp_striped = "NO"
_hp_root_grp_striped help_text "Stripe root VG disks?"
```

The expression in the next example is defining a use model. Use models are not variables; they are `true` / `false` expressions that are used to control how things are done[18]. Like variables, use models can be viewed using the **Additional** button of the **Basic** tab in the Ignite-UX GUI. When you are in the **Additional** dialog box, use models look visibly different from variables. Later in the configuration file, is the impact statement that this use model would have when it is set to `true`.

```
INIT[19] "Create /export volume" = false
```

Now you have another effects relationship, this time between a new variable `_hp_root_grp_disks` and `_hp_root_disk`. The variable `_hp_root_grp_disks` is meant to be an integer so when doing the effects relationship you add zero (`0`) to

---

[18] Unfortunately, variables cannot have effects relationships with use models or software selections.

[19] The "`init`" keyword is case insensitive, so it can appear is `init` or `INIT`. It cannot appear in mixed case such as "Init". If you do this, Ignite-UX will not recognize it as a keyword.

disk[_hp_root_disk].size to ensure that the final value is an integer. Since variables are not explicitly typed, you want to imply an integer type.

The variable _hp_root_grp_disks is also an enum; it can take one of the values from 1 up to the value of variable num_disks,[20] and it has a default initial value of 1.

```
#
# Number of disks in the root volume group
# The fist line below is just to define an "effects" relationship
# between the root disk and _hp_root_grp_disks which dependent
# upon each other in the logic to follow.
#
init _hp_root_grp_disks = (disk[_hp_root_disk].size+0) # +0 convert to int
enum _hp_root_grp_disks
_hp_root_grp_disks = { 1..num_disks }
init _hp_root_grp_disks = 1
_hp_root_grp_disks help_text "# of disks in root VG"
```

In the next example, you test on the size of the root disk and the number of disks attached to the system. If the size of the root disk is <600MB and there are only two disks, the second disk is automatically included into the root volume group (by setting _hp_root_grp_disks to be 2).

When the root disk is <600MB, the default value for _hp_disk_layout is "Whole disk (not LVM) with HFS". If there are only two disks in the system, the second disk is included and the default for _hp_disk_layout is changed to "Logical Volume Manager (LVM) with VxFS".

```
# If there are only two disks and the "_hp_root_disk" disk is small, default
to
# LVM configuration with both disks in root volume group.
disk[_hp_root_disk].size < 600Mb & num_disks == 2
{
    init _hp_disk_layout = "Logical Volume Manager (LVM) with VxFS"
    init _hp_root_grp_disks = 2
}
```

Next is another use model defined called "Create separate volumes (/usr, /var, ...)". It is used later for _hp_disk_layouts other than "Whole disk (not LVM) with HFS" only. By default, this is TRUE (and the Ignite-UX GUI does show separate volumes for /usr, /var, ... by default).

[20] A built-in variable that is a count of the number of disks discovered on the system. Refer, refer to *instl_adm*(4).

```
INIT "Create separate volumes (/usr, /var, ...)"  = TRUE
```

The first disk configuration "Whole disk (not LVM) with HFS" is protected by a test of the variable `_hp_disk_config` to see if it is equal to "Whole disk (not LVM) with HFS".

After that, the `partitioned_disk` keyword starts the definition of a whole disk layout for a system. The `partitioned_disk` disk definition must contain the following:

- A `physical_volume` definition
- An `fs_partition` definition
- A `swap_partition` definition

If you have no `fs_partition` definition that at least defines the `root` file system, the following error appears after pressing **Go!** in the Ignite-UX GUI:

```
ERROR: There is no root volume (mount point = /) defined in the configuration.
```

If you have no `swap_partition` defined, you see the following error after selecting **Go!** in the Ignite-UX GUI:

```
ERROR: There is no swap volume defined in the root volume group (or on the
root disk).
```

For the physical volume statement, you need only supply the name of the disk that is used as the root disk (the `disk[]` keyword takes a hardware path or index number and returns the name of the disk). This discussion here assumes that you are defining the boot disk as a whole disk[21].

However, if you were writing custom configurations you probably would not worry about whole disk layouts on any system running B.11.11 (you need a 2GB or less disk drive to do this on PA-RISC because of potential firmware limitations and HP-UX B.11.11 recommends at least a 4GB root disk).

```
_hp_disk_layout == "Whole disk (not LVM) with HFS"
{
    partitioned_disk
    {
        physical_volume disk[_hp_root_disk]
```

[21] You can have multiple whole disks defined in a configuration. Each whole disk is defined by different `partitioned_disk` definitions and has a different `mount_point`.

The file system partition on a partitioned disk must be HFS. It follows from the fact that the PA-RISC boot loader can only read from some types of HFS file system (large files-enabled HFS file systems cannot be booted from on PA-RISC systems). Therefore, the usage statement is set to be HFS. You set the block size and the fragment size[22] to the HFS defaults defined earlier.

Now size is set to the remaining, which takes whatever space is left over once "other" things (in this case swap at the end of the disk) have been allocated., The mount point naturally is "/" (since it is the `root` file system). Lastly, if this is a very small disk change `minfree` to be 5% (you are installing B.11.11 and there is no way you can fit B.11.11 onto a 300MB disk anyway).

```
fs_partition {
    usage = HFS
    blksize = _hp_HFS_blksize
    fragsize = _hp_HFS_fragsize
    size = remaining
    mount_point = "/"
    disk[_hp_root_disk].size < 300Mb {
        # For really small disks, tune down minfree
        # in order to gain some disk space.
        minfree = 5
    }
}
```

Now you have our primary (and presumably only) swap partition definition, you can see that it is swap from the usage. The interesting thing is the size specification. This sets the size to be at least `_hp_min_swap` but also includes any remaining space up to `_hp_pri_swap` in size. The swap space in this case attempts to get all of the space that it can (up to `_hp_pri_swap`) but does not go lower than `_hp_min_swap` while ensuring that the file system partition gets enough to meet its impacts[23] statements requirements.

```
swap_partition {
        usage = SWAP
        mount_point = "primary"
        size = _hp_min_swap | remaining | _hp_pri_swap
        }
    }
```

---

[22] These concepts are only meaningful in a HFS file system. The block size is the largest block of data a file can have allocated to it (files that can occupy a full free block will do so and the smallest amount of space that can be allocated by a file is the fragment size. In extent-based file systems these concepts are usually meaningless. For example, the smallest allocation unit is 1KB no matter what the fragment size is set to for the file system.

[23] Impact statements and software are discussed later.

```
}
```

Now you have all of the volume manager layouts. You may have noticed at the beginning of this section that LVM and VxVM both use the same ways of defining volume definitions; only the usage statement at the volume group level distinguishes them.

```
_hp_disk_layout == "VERITAS Volume Manager (VxVM) with VxFS" |
_hp_disk_layout == "Logical Volume Manager (LVM) with HFS" |
_hp_disk_layout == "Logical Volume Manager (LVM) with VxFS"
{
```

Here you are defining the variable `_hp_group_name`. However, you must first make sure that it is not visible on the **Additional** button (since you do not want anyone to be able to change it). If the value of `_hp_disk_layout` is set to be VxVM ("VERITAS Volume Manager (VxVM) with VxFS") you set `_hp_group_name` to be `rootdg`; otherwise it is LVM you are using and `vg00` is the name you are going to use[24].

The `_hp_group_name` does not actually do anything except hold the initial name of the volume group or disk group that this configuration is defining. If this variable could be changed using the **Additional** button, it would not do anything since it is only a temporary variable to hold the name[25].

```
_hp_group_name visible_if FALSE


 (_hp_disk_layout == "VERITAS Volume Manager (VxVM) with VxFS") {
   _hp_group_name = "rootdg"
} else {
   _hp_group_name = "vg00"
    }
```

The temporary variable is necessary to give the volume group a name. You do this by assigning the name to a temporary variable and then starting your volume group definition with that name.

A volume group definition must have the following things in it:

- volume group attributes
- one or more `physical_volume` definitions (including "group"[26] definitions that also include more physical volume definitions)
- one or more (usually more) `logical_volume` definitions.

```
    volume_group _hp_group_name
```

[24] VxVM 3.5 and earlier requires the root volume group be named `rootdg`. This is enforced by Ignite-UX.
[25] To change the name of a volume group you use the Ignite-UX GUI to select the **File system** tab then the **Additional Tasks** button, and click **Group Parameters**. From the **Group Parameters** dialog box, select the name of the group to change from the selection list, change the group name, press modify, and then click **Ok**.
[26] Defining physical volume groups is discussed later when discussing custom configuration.

```
    {
```

The `physical_volume` definition places the first disks (whose number is determined by `_hp_root_grp_disks`) returned by the `disk[]` construct into the root volume group[27]. When Ignite-UX encounters the first volume group that uses `*=<index>` it attempts to return the preset root disk as the first disk. This causes things to look as though the disk in `_hp_root_disk` (initialized from `_hp_primary_path`) was always placed into the list of physical volumes put into the root volume group and disk group.

```
physical_volume disk[*=_hp_root_grp_disks]
```

Next is a test to choose the volume manager. Currently the only choice that enables VxVM is "VERITAS Volume Manager (VxVM) with VxFS". All of the other choices use LVM.

```
(_hp_disk_layout == "VERITAS Volume Manager (VxVM) with VxFS") {
    usage = VxVM
} else {
    usage = LVM
}
```

All the volume group attributes such as `max_physical_extents` are optional and only affect LVM volume groups. If you do not give them, they take on the defaults of the `vgcreate` command.

```
# In order to accommodate adding 9GB disks now or in the
# future, set the default max_physical_extents large
# enough to handle it (based on 4Mb extent size).
# Note that this will be increased by IUX automatically
# for disks larger than 9GB.
max_physical_extents = 2500
```

Very large disks are the most common disks on new systems with ≥140GB disks considered large, 36/72GB disks average, and 18GB disks small. These tests adjust the physical extent size for the LVM volume group being defined and ensures that the complete disk can be used with the disks. It is only based upon the size of the root disk, not the size of every disk that might be included. This might influence the amount of usable space in the volume group. Additionally, Ignite-UX examines all the disks being used and selects the LVM parameters that work for that entire disk group. The setting of physical extent size here is just a first approximation; Ignite-UX can change these values at a later stage.

---

[27] This is evaluated once. After that, changes made via the Ignite-UX GUI affect which physical volumes is used in the root volume group.

In a custom configuration, it might be better to scale both `max_physical_extents` and `physical_extent_size` based upon the disk size.

```
#
# For very large disks, the root group VGRA meta-data area
# will outgrow its bounds and trigger an IUX sanity check
# error.  So for root disks large enough, set the physical
# extent size.  Note that this only handles the root disk, but
# any disk in the root VG could trigger the error...
#
 (disk[_hp_root_disk].size > 21504MB) # >21GB
{
    physical_extent_size=8
}
 (disk[_hp_root_disk].size > 44032MB) # >43GB
{
    physical_extent_size=16
}
(disk[_hp_root_disk].size > 84992MB) # >83GB
{
    physical_extent_size=32
}
(disk[_hp_root_disk].size > 179200MB) # >175GB
{
    physical_extent_size=64
}
```

Next are the definitions of the logical volumes in the configuration. The first one is, of course, the `root` file system (note the `mount_point` definition). Depending on the value of the disk layout, the file system type is either HFS or VxFS.

Further, you have a test for a use model "Create separate volumes (/usr, , ...)". If there are separate file systems, then the default size for the `root` file system is 140MB if the root disk is less than 8192MB, or 200MB if it is greater than or equal to 8192MB.

If you are not expected to create separate logical volumes for file systems like `/usr` and `/var` then the sizes are the remaining space up to 1200MB if the size of the disk is less than 8192MB or the remaining space up to 1200MB plus the size of `_hp_pri_swap`[28].

[28] HP-UX B.11.11 may not actually fit into the amount of space defined here; the impacts statements for software defined may also force the sizes of file systems to be adjusted.

The root file system cannot have bad block relocation on and must be contiguous; therefore contiguous_allocation is set to true, and bad_block_relocation is set to false.

```
logical_volume {
    mount_point = "/"
    _hp_disk_layout == "Logical Volume Manager (LVM) with HFS" {
         usage = HFS
         blksize = _hp_HFS_blksize
         fragsize = _hp_HFS_fragsize
    } else {
         usage = VxFS
         blksize = _hp_VxFS_blksize
    }
    "Create separate volumes (/usr, /var, ...)" {
         disk[_hp_root_disk].size < 8192Mb {
              size = 140Mb
         } else {
              size = 200Mb
         }
    } else {
         disk[_hp_root_disk].size < 8192Mb {
              size = remaining | 1200Mb
         } else {
              size = remaining | 1200Mb + _hp_pri_swap
         }
    }
    contiguous_allocation = true
    bad_block_relocate    = false
}
```

Next, you define primary swap by setting the usage to be swap and dump. The mount_point is set to "primary". The vgdisk[0] value enables you to specify that this logical volume be placed onto the first disk in the root volume group. Refer to *instl_adm*(4) for more information on the vgdisk keyword.

Next are the usual settings for primary swap of contiguous allocation and disabled bad block relocation. The size must be at least _hp_min_swap and up to _hp_pri_swap depending on the size of the disk.

```
logical_volume {
    usage = SWAP_DUMP
    mount_point = "primary"
    # Map it to the root disk to ensure it is considered primary
    # without this the secondary swap gets sorted first and considered
```

```
    # primary.
    vgdisk[0]
    contiguous_allocation = true
    bad_block_relocate    = false
    size = _hp_min_swap | remaining | _hp_pri_swap
}
```

You looked at `_hp_sec_swap` earlier in the configuration file to see how it was defined. The only place you can give it a value is by using the **Additional** button on the **Basic** tab in the Ignite-UX GUI. You can manually define secondary swap using the **File system** tab, but you must manually select the disk to avoid it being on the same disk as primary swap. The configuration enables you to have secondary swap/dump. You only have to care about its size, not placement.

So if the variable `_hp_sec_swap` is >0MB, swap/dump space is created. If there is more than one disk in the root volume group, this swap/dump space is created on the second disk. If you provide equal sized primary and secondary swap, it creates an interleaved swap setup.

Also, you set the options required for this swap space to also be used as a dump space (contiguous allocation and bad block relocation off), and lastly you set the size to be `_hp_sec_swap`. Secondary swap, if not used for dump, does not require contiguous allocation and `bad_block` relocation to be `_relocate` set to `true`.

```
_hp_sec_swap > 0Mb {
    logical_volume {
        usage = SWAP_DUMP
        mount_point = "secondary"
        (num_vgdisks > 1)
        {
            # This maps the secondary swap space to
            # the second disk when one is defined.
            vgdisk[1]
        }
        contiguous_allocation = true  # allows use as dump
        bad_block_relocate    = false # allows use as dump
        size = _hp_sec_swap
    }
}
```

Next `/stand` is defined. Of course, it must be HFS[29]. The block and fragment sizes are initialized from constants defined earlier. If you have a disk less than 2050MB, `/stand`

[29] The secondary loader in HP-UX B.11.11 cannot read VxFS file systems.

defaults to 84MB. A disk greater than or equal to 2050MB but less than 3075MB is 112MB in size; otherwise you get a 300MB `/stand` file system by default.

```
logical_volume {
    mount_point = "/stand"
    # /stand must be HFS!
    usage = HFS
    blksize = _hp_HFS_blksize
    fragsize = _hp_HFS_fragsize
    # The /stand volume needs to be able to hold at least 5 kernels,
    # for various debugging reasons.  The typical 11i kernel is
    # about 18Mb.  The debug IA kernel is 42Mb.  The smallest /stand
    # area needs to be at least twice the largest possible kernel.
    # Unless a user tries to put on more than the "normal" amount of
    # software on a system with a large root disk, the default /stand
    # directory will be 300Mb.  To see the use of the size option,
    # see the man page for "instl_adm".
    disk[_hp_root_disk].size < 3075Mb {
          disk[_hp_root_disk].size < 2050Mb {
                # Most 2G drives are actually bigger than exactly 2G
                size = 84Mb
          } else {
                size = 112Mb
          }
    } else {
          size = 300Mb
    }
    contiguous_allocation = true
    bad_block_relocate    = false
}
```

300MB is the default for large disk configurations because additional DLKMs[30] require more space in `/stand`.

Now at this stage you may be wondering why you had to make sure that you set up those logical volumes with certain values for contiguous allocation and bad block relocation. Table 3 is a brief summary:

Table 3

Name	Contiguous Allocation	Bad Block Relocation[31]	Reason

[30] Dynamically loadable kernel modules

/	TRUE	FALSE	The root file system must be usable before the volume manager is active to enable this. It cannot allow the volume manager to relocate blocks and it must be contiguous on a disk.
Primary swap	TRUE	FALSE	Primary swap is activated before the volume manager is active, so it requires the same options as the root file system.
Dump space	TRUE	FALSE	A dump is written when no volume manager is active; because of this, it must be contiguous, and cannot have bad blocks relocated by the volume manager.
Secondary swap	N/A	N/A	Secondary swap has no requirements unless it is also a dump space; in which case, it has the dump space requirements.
/stand	TRUE	FALSE	The /stand volume is read by the boot loader. The boot loader knows nothing about volume managers and currently only understands HFS[32] file systems, so the file system must be contiguous and cannot have bad block relocation enabled.

Note:
On HP-UX B.11.11, it is assumed that all IODC (I/O Dependent Code) in firmware can perform operations with up to 4GB offsets[33]. For some I/O controllers this limits the maximum offset for dump. Newer systems have block-based I/O implemented in IODC. This allows the offset of an I/O operation to be given in blocks not bytes. This change dramatically increases the amount allowed for offset of dump spaces on most new I/O interfaces. Itanium®-based systems do not have a 4GB offset limit for dump since they all support block-based dump. In these cases, the size is limited to the size of the disk itself.

If the option to create separate volumes is true (the default), then the following very long block of definitions is examined:

```
"Create separate volumes (/usr, /var, ...)" {
```

Next is a definition for /usr. The file system has large files enabled, and if the disk layout is "Logical Volume Manager (LVM) with HFS" then a HFS file system is created with the

[31] This is bad block relocation performed by a volume manager not automatic relocation performed by the disk drives.
[32] Itanium®-based systems may have /stand as a VxFS filesystem. The HFS restriction is applicable only to PA-RISC systems.
[33] Various combinations of 2GB and 4GB offsets (depending on the I/O interface) were allowed in older releases of Ignite-UX (before B.3.0 and in all of the A.*x.x* versions).

default block and fragment sizes set up earlier. Otherwise, it is a VxFS file system with the default VxFS block size (note that the fragment size is irrelevant[34] for VxFS).

```
logical_volume {
    mount_point = "/usr"
    largefiles = true
    _hp_disk_layout == "Logical Volume Manager (LVM) with HFS" {
          usage = HFS
          blksize = _hp_HFS_blksize
          fragsize = _hp_HFS_fragsize
    } else {
          usage = VxFS
          blksize = _hp_VxFS_blksize
    }
```

Now if the size of the root disk is less than 1800MB, the size of the volume is at least 332MB, after all other volumes have had their sizes allocated. The size of `/usr` can be increased to have up to 20% free space[35] (if available). The impact of disk space requirements for software being installed that affects `/usr` may force the size to be higher.

```
disk[_hp_root_disk].size < 1800Mb {
    size = 332Mb | remaining | 20% free
} else {
```

If the root disk size is >3072 MB, the size of /usr is 500MB. Once all other volumes have had their space allocated, the size can be increased to ensure that there is 20% free space (noting the issues discussed previously can impact the size requirements). If the root disk size is between 1800MB and 3072MB, then the minimum size is 400MB.

```
    disk[_hp_root_disk].size > 3072MB {
          size = 500Mb | remaining | 20% free
    } else {
          size = 400Mb | remaining | 20% free
    }
}
```

[34] VxFS is an extent-based file system and allocates available space to a file. Fragments can be as low as 1KB but are typically larger. There is no way to control fragment size in a VxFS file system.
[35] This amount does not include the amount of space that can be added by the `_hp_addnl_fs_free_pct` configuration item.

If the configuration item _hp_root_grp_striped is set to "YES" then you set up striping. The "stripes = *" set the number of stripes equal to the number of disks in the volume group. The stripe size _hp_FS_stripe_size was set up earlier as 64KB.

```
_hp_root_grp_striped == "YES"  {
    stripes = *
    stripe_size = _hp_FS_stripe_size
    }
}
```

The definition of /opt is very similar to the definition of /usr. For this volume and the others defined by setting "Create separate volumes (/usr, /var, ...)" to true, you only look at the differences rather than the similarities.

The only difference between /usr and /opt is the size definition: rather than giving a percentage free, there is a specific size requirement for free space. For example, in the first size definition, when the size of the root disk is <1800MB, /opt is a minimum of 100MB. After all space has been allocated, you take from the remaining up to an extra 100MB and place it into /opt[36].

```
logical_volume {
    mount_point = "/opt"
    largefiles = true
    _hp_disk_layout == "Logical Volume Manager (LVM) with HFS" {
          usage = HFS
          blksize = _hp_HFS_blksize
          fragsize = _hp_HFS_fragsize
    } else {
          usage = VxFS
          blksize = _hp_VxFS_blksize
    }
          disk[_hp_root_disk].size < 1800Mb {
          size = 100Mb | remaining | 100Mb free
    } else {
          disk[_hp_root_disk].size > 3072MB {
          size = 100Mb | remaining | 252Mb free
    } else {
          size = 100Mb | remaining | 152Mb free
```

[36] Again, the impact of software being installed can increase the size of this volume, and the _hp_addnl_fs_free_pct configuration item can increase the final size of the volume (if free space is still left unallocated).

```
        }
    }
    _hp_root_grp_striped == "YES"  {
        stripes = *
        stripe_size = _hp_FS_stripe_size
    }
}
```

If the size of the root disk is greater than or equal to 8192MB, the final size can be 500MB plus the size of `_hp_pri_swap`. The value is based on primary swap being dump and some space is required in `/var/adm/crash` to save crash dumps. This assumption is invalid if you define a separate volume for `/var/adm/crash` using the Ignite-UX GUI.[37]

```
logical_volume {
    mount_point = "/var"
    largefiles = true
    _hp_disk_layout == "Logical Volume Manager (LVM) with HFS" {
        # /var needs lots of inodes due to SD's IPD
        nbpi = 2048
        usage = HFS
        blksize = _hp_HFS_blksize
        fragsize = _hp_HFS_fragsize
    } else {
        usage = VxFS
        blksize = _hp_VxFS_blksize
    }
    disk[_hp_root_disk].size < 8192Mb {
        size = 72Mb | remaining | 500Mb
    } else {
        size = 72Mb | remaining | 500Mb + _hp_pri_swap
    }
    _hp_root_grp_striped == "YES"  {
        stripes = *
        stripe_size = _hp_FS_stripe_size
    }
}
```

[37] The size requirement is up to 500MB + `_hp_pri_swap`. Depending on free disk space, there may not be enough to increase the size of a file system this large. During an interactive installation, the size of `/var` can be manually tuned down to a more appropriate value.

The definition of /tmp is a simpler case than the other volumes. In this case, the size is a fixed value. It is unlikely that any impacts caused by installing software would affect /tmp.

```
logical_volume {
    mount_point = "/tmp"
    largefiles = true
    _hp_disk_layout == "Logical Volume Manager (LVM) with HFS" {
         nbpi = 2048
         usage = HFS
         blksize = _hp_HFS_blksize
         fragsize = _hp_HFS_fragsize
    } else {
         usage = VxFS
         blksize = _hp_VxFS_blksize
    }
         disk[_hp_root_disk].size < 8192Mb {
         size = 64Mb
    } else {
         size = 200Mb
    }
    _hp_root_grp_striped == "YES"  {
         stripes = *
         stripe_size = _hp_FS_stripe_size
         }
    }
} # "Create separate volumes (/usr, /var, ...)"
```

Although it is rare to see an /export volume created, you can enable it by using the **Additional** button on the **Basic** tab in the Ignite-UX GUI.

```
    "Create /export volume" {
         logical_volume {
              mount_point = "/export"
              largefiles = true
              _hp_disk_layout == "Logical Volume Manager (LVM) with HFS" {
                   usage = HFS
                   blksize = _hp_HFS_blksize
                   fragsize = _hp_HFS_fragsize
              } else {
```

```
                usage = VxFS
                blksize = _hp_VxFS_blksize
            }
            size = remaining | 100Mb
            _hp_root_grp_striped == "YES"  {
                stripes = *
                stripe_size = _hp_FS_stripe_size
            }
        }
    }
```

The /home volume always gets defined and has a rather small size with no minimum, going up to 20MB.

```
logical_volume {
    mount_point = "/home"
    largefiles = true
    _hp_disk_layout == "Logical Volume Manager (LVM) with HFS" {
        usage = HFS
        blksize = _hp_HFS_blksize
        fragsize = _hp_HFS_fragsize
    } else {
        usage = VxFS
        blksize = _hp_VxFS_blksize
    }
    size = remaining | 20Mb
    _hp_root_grp_striped == "YES"  {
        stripes = *
            stripe_size = _hp_FS_stripe_size
            }
        }
    } # root group
} # Volume Manager disk layout.
```

The following special variables are used by Ignite-UX to control aspects of the installation or recovery. The "Special variables" section describes all of the special variables in detail.

```
# Set the variables that are recognized by the UI as invisible
# so that they don't show on the "Additional" screen:
_hp_locale           visible_if false
_hp_cfg_detail_level visible_if false
```

```
_hp_pri_swap         visible_if false
_hp_min_swap         visible_if false
_hp_disk_layout      visible_if false
_hp_default_cur_lan_dev   visible_if false
_hp_default_final_lan_dev visible_if false
_hp_keyboard         visible_if false
_hp_root_disk        visible_if false
_hp_boot_dev_path    visible_if false
```

If `_hp_disk_layout` is one of the volume manager values then you allow some things to become visible in the **Additional** button from the **Basic** tab in the Ignite-UX GUI. Some of these control some of the configuration that you have already looked at. They are only applicable if you are using a volume manager, so the `else` construct makes sure that none of them can be seen on the **Additional** button.

```
_hp_disk_layout == "VERITAS Volume Manager (VxVM) with VxFS" |
_hp_disk_layout == "Logical Volume Manager (LVM) with HFS" |
_hp_disk_layout == "Logical Volume Manager (LVM) with VxFS"
{
    "Create /export volume"                     visible_if true
    "Create separate volumes (/usr, /var, ...)" visible_if true
    _hp_sec_swap         visible_if true
    _hp_root_grp_striped visible_if (_hp_root_grp_disks > 1)
    _hp_root_grp_disks   visible_if true
}
else
{
    "Create /export volume"                     visible_if false
    "Create separate volumes (/usr, /var, ...)" visible_if false
    _hp_sec_swap         visible_if false
    _hp_root_grp_striped visible_if false
    _hp_root_grp_disks   visible_if false
}
```

The following `sw_sel` definition adds some impacts and enables the addition of the impacts to the file systems to take into account file system usage caused by the installation. When the `visible_if` variable is set to `false` the `sw_sel` definition does not appear in the Ignite-UX GUI; this variable is preset to `true` so the impacts always affect the installation.

```
#
# In order to more accurately specify the software impact to
```

```
# specific volumes that normally do not have files loaded
# directly, but get created by scripts, we specify some
# typical sizes here so the disk space is accounted for.
#
init sw_sel "impact by scripts" {
    sw_source = "core"
    visible_if = false
    impacts = "/var"   60Mb
    impacts = "/stand" 20Mb
} = TRUE
```

The following configuration statements add a selection to the **Additional** button that enables the user to control the use of `autoboot`. Ignite-UX normally runs no matter what the `autoboot` flag is set to (refer to *`setboot`*`(1M)`). It forces it on if it was off, then later turns it back off for the final reboot when the system would normally come back. This configuration enables the user to manually boot the system when `autoboot` is disabled.

Ignite-UX normally forces the system to perform the first reboot automatically when rebooting from the installation kernel to the newly created or recovered kernel. This reboot is forced no matter what the `autoboot` flag is set to (refer to *`setboot`*`(1M)`). Ignite-UX forces it on if it was off; prior to the final reboot Ignite-UX returns the `autoboot` flag to off. If `_hp_force_autoboot` is set to `NO` and the system's `autoboot` flag is not set to on, the system does not perform a reboot.

Important:
If you disable `autoboot` and then install a system, the system never executes a final reboot without manual intervention. HP does not support booting into any other mode than the default multi-user run level at this point. The installation or recovery session has not yet completed and some steps must still be performed by Ignite-UX. Booting into anything other than the default run level does not allow the installation or recovery to complete. This is the intended behavior of Ignite-UX and not a flaw. In general (except when investigating Ignite-UX issues on advice from HP Support), you should never change the value of the `_hp_force_autoboot` variable.

```
#
#  Variable to give user control over the use of setboot. When the variable
#  is YES, autoboot is set before the first reboot and restored after the
#  first reboot. When the variable is NO, autoboot is never changed.
#
enum _hp_force_autoboot
_hp_force_autoboot = { "YES", "NO" }
init _hp_force_autoboot = "YES"
```

```
_hp_force_autoboot help_text "Force Ignite-UX autoboot?"
```

The following example enables you to set, using the **Additional** button, whether the final system uses DHCP to obtain an IP address and hostname once it is installed or recovered. Ignite-UX uses DHCP to get initial networking information (you must disable DHCP [`disable_dhcp=true`] in the installation file system to stop a system booting over the network from using DHCP to obtain an IP address and hostname).

```
#
#  Variable to give user control over DHCP.
#  If YES, then DHCP is turned off on the system.
#
enum _disable_dhcp
_disable_dhcp = { "YES", "NO" }
init _disable_dhcp = "NO"
_disable_dhcp help_text "Disable DHCP?"
_disable_dhcp == "YES"  {
    DISABLE_DHCP=TRUE
}
```

In the next example, you can configure the value of `_hp_addnl_fs_free_pct`. The variable is defined as a list of values that can be selected from the **Additional** button. If an `enum` had been placed before the `_hp_addnl_fs_free_pct` you would only be able to select a value from 0 to 10. Instead, you can select a value from 0 to 10 or enter another value when changing the variable using the **Additional** button.

```
#
#  JAGae82704 fix. Variable to give user control over the amount of
#  free space left over in a logical volume. It defaults to 10.  The
#  range provided if you click on the button is from 0-10, but a larger
#  value can be entered directly in the field if desired.
#
_hp_addnl_fs_free_pct = { 0,1,2,3,4,5,6,7,8,9,10 }
init _hp_addnl_fs_free_pct = 10
_hp_addnl_fs_free_pct help_text "Additional free space %"
```

The last part of configuration accomplishes several things:

1. Sets the `os_release` attribute for the `swinstall` command line.
2. Makes sure that `_hp_os_bitness` is set and set correctly (it should be set by the configuration file that defines the core operating system depot or archive `sw_sel` clause).
3. Makes sure that the system can run the bitness of HP-UX.
4. Sets the operating system Name attribute for the `swinstall` command line.

```
#####################################################################
# Ensure that the swinstall command line used contains the correct pose_as
# attributes for the type of system being installed.
#
sd_command_line += " -x os_release=" +[38]${release}


(_hp_os_bitness == "64")
{
    (!can_run_64bit)
    {
          ERROR += "This system model: \"" +${model}+ "\" is not supported for
running 64bit HP-UX, you must select the 32bit selection"
    }
    sd_command_line += " -x os_name=HP-UX:64 "
}
else
{
    (_hp_os_bitness == "32")
    {
          (!can_run_32bit)
          {
          ERROR += "This system model: \"" +${model}+ "\" is not supported for
running 32bit HP-UX, you must select the 64bit selection"
          }
          sd_command_line += " -x os_name=HP-UX:32 "
    }
    else
    {
          ERROR += "The _hp_os_bitness variable is not set to \"32\" or \"64\".
This variable must be set in the config file that supplies the core archive or
depot.  If using an archive, make sure you start with the core11.cfg example
config file.  When using a depot, make_config will set this automatically."
    }
}
_hp_os_bitness visible_if false
```

You should exercise care when setting the value of `sd_command_line` in any other configuration file because the `os_name` and `os_release` variables are required to install

[38] Concatenates two strings together using the + operator. For more information, refer to *instl_adm*(4).

software correctly during an initial installation. Using = rather than += to add an option to sd_command_line removes any current settings from sd_command_line.

Special variables

Ignite-UX uses the following special variables to control aspects of installation or recovery, or to modify the state of the final system. The descriptions of these variables come from the *instl_adm*(4) manpage, with further explanation provided where possible. If the variable appears directly in or impacts the Ignite-UX GUI, this information is also explained.

_hp_locale

The geocustoms user interface (UI) sets this string variable to the language locale that the user has chosen. If set to the string ASK_AT_FIRST_BOOT then the *geocustoms*(1M) application is invoked when the installation is complete, allowing the user to pick the desired language. (The geocustoms application is not available prior to HP-UX 10.30). Setting the value to SET_NULL_LOCALE leaves the system at default with no LANG variable set, which causes the commands to use the internal messages that provide better performance.

If _hp_locale is not set, it is set internally to the first value of the locale keyword for a selected sw_sel. If multiple selected sw_sels contain locale keywords, the _hp_locale variable is set to the first value of the locale of the first selected sw_sel.

The geocustoms[39] application is used to manage the default locale on a system. If a locale is not important, do not set one. If you set _hp_locale, set it to a locale that exists on the final installed system, it should be visible in the output of locale -a.

You can run the geocustoms command at any time to manage the languages installed on your system (for example, you can remove some or change the default locale). You cannot use the geocustoms command to install non-existent locales back again.

The geocustoms program modifies /etc/rc.config.d/LANG to set a locale. The value of SET_NULL_LOCALE for _hp_locale leaves /etc/rc.config.d/LANG with no locale set.

The geocustoms program is easy to use because it provides a full-screen menu-driven interface for managing locale support. It does have command line options if you prefer not to use the user interface.

The standard configuration files set the variable _hp_locale visible_if false to prevent this variable from being visible or changeable when using the **Additional** button in the Ignite-UX GUI. The locale is chosen using the **Languages** button on the **Basic** tab in the Ignite-UX GUI.

_hp_cfg_detail_level

This internal string variable is fundamental to the configuration file management done by Ignite-UX. It is used primarily in client-specific configuration files. It contains a list of option characters that represent which aspects of the configuration file are modified by the

[39] The geocustoms command is deprecated and is planned for obsolescence in a future release of HP-UX.

user interface. This represents the areas of information that the configuration file, written by the UI, contains.

The Ignite-UX process uses this information in case it needs to rewrite the configuration file. The configuration file is rewritten both by the user interface and by the client installation process. In these cases, the process uses `_hp_cfg_detail_level` to determine how much information is represented by the configuration file and so it can write out just the minimal amount of information and let the rest be supplied by the other configuration files in the `INDEX` file.

The option characters recognized in the `_hp_cfg_detail_level` string that is found in a client-specific `config` file are as follows:

`-i INDEX cfg` clause selection (file contains the line telling which `INDEX` file entry should be used).

`-v` Variable and use model settings.

`-s` Software selection settings.

`-S` Modified software selection definitions.

`-r` Modified software source definitions (depot information).

`-f` Modified file system information.

`-p` System identity information.

`-T` The `post_config_script` selections settings.

`-h` Hardware control information (`hw_instance_num` statements).

`-l` Other control information (`global mod_kernel` statements for example).

For example, the line `_hp_cfg_detail_level="ivsp"` in a client configuration file would indicate that the file contains information about which `cfg INDEX` selection is to be used, as well as the variable settings, software selection settings, and system parameters.

The user interface rewrites the `config` and `config.full` files in the per-client directories on the Ignite-UX server for network installation and recovery, or in the `/var/opt/ignite/local` directory for local media installation and recovery.

For network recoveries and installations, if you manually modify a file that Ignite-UX can modify, you must be careful that the `_hp_cfg_detail_level` contains the appropriate level of detail since it tells Ignite-UX what information is represented in the file that is what should be written back out into the file when it is rewritten by anything. If you add information that is not listed in the variable `_hp_cfg_detail_level`, it can be read, but then, the unlisted information is not written back out again even if other information in the file is modified and the file needs to be rewritten.

The standard configuration files use `_hp_cfg_detail_level visible_if false` to prevent this variable from being visible or changeable when using the **Additional** button on the **Basic** tab in the Ignite-UX GUI.

_hp_pri_swap

The _hp_pri_swap is the integer variable that is set by the Ignite-UX GUI to indicate how much swap the user desires to have allocated on the root disk/volume-group. The default LVM primary swap volume is defined as a size range with _hp_pri_swap being the desired size and _hp_min_swap being the minimum size.

On the **Basic** tab in the Ignite-UX GUI, if you change the value of the **Root Swap (MB)...** field you have changed the value of _hp_pri_swap.

Important:
If you navigate to the **File system** tab and select the primary swap volume to change its size, it does not reflect the change into the value on the **Basic** tab. That value remains the size as the primary swap. However, if you return to the **Basic** tab and change the field labeled **Root Swap (MB)...** at all, it changes the primary swap size on the **File system** tab, so exercise caution.

The standard configuration files use the variable _hp_pri_swap visible_if false to prevent this variable from being visible or changeable from the **Additional** button in the Ignite-UX GUI.

_hp_min_swap

The _hp_min_swap is the integer variable that is used as the minimum size the primary swap volume can be reduced to if there is not enough disk space for all other volumes. This value defaults to an amount that should allow the system to boot and run HP-UX.

On the **File system** tab in the Ignite-UX GUI, if you select primary swap in the list of file systems you see an immediate change to the fields below the section list; the field labeled Min: is the value of _hp_min_swap.

The standard configuration files use _hp_min_swap visible_if false to prevent this variable from being visible or changeable when using the **Additional** button in the Ignite-UX GUI.

_hp_disk_layout

The _hp_disk_layout string variable is set by the Ignite-UX GUI to indicate which disk layout (LVM, whole-disk, etc) that you have selected. As configurations are saved, the list of values that it can have is increased to contain any modified layouts.

There are ways that you should and should not define this variable. The following is the correct way to define the _hp_disk_layout variable; this assumes that _hp_root_disk has already been set.

```
_hp_disk_layout= { "N4000 with 9Gb disk", "N4000 with 18Gb disk",
"N4000 with 36Gb disk", "N4000 with 72Gb disk" }
(model ~ "9000/N4000")
{
    (disk[_hp_root_disk].size<10000Mb)
```

```
            {
                  init _hp_disk_layout=" N4000 with 9Gb disk"
            }
      else
            {
                  (disk[_hp_root_disk].size<20000Mb)
                  {
                        init _hp_disk_layout=" N4000 with 18Gb disk"
                  }
      else
            {
                  (disk[_hp_root_disk].size<400000Mb)
                  {
                        init _hp_disk_layout=" N4000 with 36Gb disk"
                  }
      else
            {
                  init _hp_disk_layout=" N4000 with 72Gb disk"
            }
        }
    }
}
```

The preceding configuration correctly defines the settings for `_hp_disk_layout` to enable the Ignite-UX GUI to modify the disk configuration. If you want to set `_hp_disk_layout` so that you cannot make any changes using the Ignite-UX GUI, follow the next example. In this configuration, you cannot change the disk layout after it has been selected by these tests:

```
_hp_disk_layout= { "N4000 with 9Gb disk", "N4000 with 18Gb disk",
"N4000 with 36Gb disk", "N4000 with 72Gb disk" }
(model ~ "9000/N4000")
{
    (disk[_hp_root_disk].size<10000Mb)
          {
                _hp_disk_layout="N4000 with 9Gb disk"
          }
    else
          {
                (disk[_hp_root_disk].size<20000Mb)
                {
```

```
                    _hp_disk_layout="N4000 with 18Gb disk"
               }
     else
          {
               (disk[_hp_root_disk].size<400000Mb)
               {
                    _hp_disk_layout="N4000 with 36Gb disk"
               }
     else
          {
               _hp_disk_layout="N4000 with 72Gb disk"
          }
       }
     }
}
```

The difference between the two sets of configuration is that the `init` keyword is missing from the second set of configuration statements when `_hp_disk_layout` is set to be a specific value. The *instl_adm*(4) manpage states:

```
init variable=value
    Preceding the assignment with the init keyword means that the
    variable is to be initialized to the given value, but the user
    interface is allowed to alter the value later.

variable=value
    When the init keyword is not used, then the variable cannot be
    changed by the user interface.  This type of assignment is not
    recommended for "visible" variables.
```

Therefore, the `init` keyword must proceed the value that `_hp_disk_layout` is set to; otherwise the Ignite-UX GUI cannot change it. When you modify anything using the Ignite-UX GUI, it wants to add a new value to the list that `_hp_disk_layout` can have and set it to the value of `_hp_disk_layout`. Unless `init` is used to give `_hp_disk_layout` its initial value, the Ignite-UX GUI cannot change it and all modifications to the disk layout are lost.

If you have an Ignite-UX server and a client that has been added or installed recently, you can test this as follows:

1. Enter:
   `cd /var/opt/ignite/clients`
   `itool -m pull -d <client>`
2. Change the type of disk layout by selecting the **File system** tab and choosing the correct layout from the list.

3. Quit the Ignite-UX GUI. You can choose any option on the exit screen; none of the options reboots or halts the system.
4. Edit `/var/opt/ignite/clients/<MAC>/config` and remove `init` from the front of the `_hp_disk_layout` variable.
5. Save the file.
6. Rerun the Ignite-UX GUI using the `itool` command in Step 1. If you change the file system layout, it reverts to the original value.

Important:
Only perform this test with one of the standard unmodified layouts. Do *not* perform this test with a modified disk layout. Removing `init` prevents you from modifying anything to do with the file system.

The values in `_hp_disk_layout` are only indirectly visible. The list of values in the `_hp_disk_layout` variable is shown on the **Basic** tab in the Ignite-UX GUI in the **File System:** list. The use of `init` indirectly affects how `_hp_disk_layout` works.

The standard configuration files use the variable `_hp_disk_layout visible_if false` to prevent this variable from being visible or changeable using the **Additional** button in the Ignite-UX GUI.

As defined, the preceding configuration replaces any other values assigned to `_hp_disk_layout`. If you want to retain any currently defined disk layouts, you need to start the preceding examples with the following:

```
_hp_disk_layout+= { "N4000 with 9Gb disk", "N4000 with 18Gb disk",
"N4000 with 36Gb disk", "N4000 with 72Gb disk" }
```

The **+=** operator adds to current values rather than replacing any existing values.

_hp_default_cur_lan_dev

The `_hp_default_cur_lan_dev` string variable is set to the LAN device that is enabled during the Ignite-UX process. It is used when a LAN device is omitted from keywords that can accept a LAN interface specifier. This defaults to the interface that you picked during the install; in the non-interactive case, it defaults to the LAN device the system booted from or to lan0 if not a network boot.

Therefore, a configuration that looks like this:

```
( lan[].driver ~ "btlan" ) {
…
}
```

With this variable, the preceding configuration is really evaluated as:

```
( lan[_hp_default_cur_lan_dev].driver ~ "btlan" ) {
…
}
```

The standard configuration files use `_hp_default_cur_lan_dev visible_if false` to prevent this variable from being visible or changeable using the **Additional** button in the Ignite-UX GUI.

_hp_default_final_lan_dev

The `_hp_default_final_lan_dev` string variable is similar to `_hp_default_cur_lan_dev`, but is used when network information is specified by using the final keyword. It defaults to `_hp_default_cur_lan_dev` if not set. The IP address associated with the LAN device specified by this variable is used for the entry in the `/etc/hosts` file for the system's hostname.

That means that if you have the following partial configuration:

```
final ip_addr[]="15.30.129.47"
final netmask[]="255.255.255.0"
```

The IP address information is applied to the LAN interface defined by the variable `_hp_default_final_lan_dev`, and this IP address is the one associated with the name of the host in the `/etc/hosts` file. If the value of the variable `_hp_default_final_lan_dev` is not set explicitly, it defaults to the value of the variable `_hp_default_cur_lan_dev`.

The standard configuration files use `_hp_default_final_lan_dev visible_if false` to prevent this variable from being visible or changeable using the **Additional** button in the Ignite-UX GUI.

_hp_keyboard

The `_hp_keyword` string variable is set by the Ignite-UX GUI to the keyboard language and mapping desired. The `kbdlang` keyword that was used for this in past releases is equivalent to this variable. Setting this variable in the `INSTALLFS` file (using `instl_adm`) prevents the system from prompting for a keyboard type during the install. This information is stored in `/etc/kbdlang` on the final system.

If `kbdlang` is not set, it causes Ignite-UX to run the following command on workstations and servers that have graphics consoles:

```
# /sbin/itemap -i -L -w /etc/kbdlang
```

Not all servers support graphics consoles. If you run the `itemap` command on a system that does not have the required hardware installed, the following message appears when run with the `-v` option:

```
# itemap -i -L -v -w /etc/kbdlang
```

```
No framebuffer device.
```

The `-L` option causes `itemap` to load the appropriate keymap for non-PS2 keyboards into the Internal Terminal Emulator (ITE)[40] and if a PS2 keyboard is found, the command will interactively prompt for the type of keyboard. The list is created by showing all of the keyboards found in the file `/etc/X11/XHPKeymaps` that start with `PS2_DIN`. A list of possible values that this variable can take is generated with the following command:

```
# keymap_ed -l | awk ' $5 ~ "^PS2_DIN" { print $5 } '
```

The X11 patches that modify `/etc/X11/XHPKeymaps` may add or remove keyboards from this list (although not very often) so the list is dependent on X graphics patch revisions. If you need to inspect the `XHPkeymaps` files from archives or other sources, the `keymap_ed` command, with the `-k` option, enables you to specify an alternate file to read. Refer to *`keymap_ed`*`(1)` for more information.

The standard configuration files use `_hp_keyboard visible_if false` to prevent this variable from being visible or changeable using the **Additional** button in the Ignite-UX GUI. It can be set for clients with required hardware from the **Keyboard** button on the **Basic** tab. If the hardware is not present, the only choice presented is "Not Applicable". If "Not Applicable" is chosen or left as a default, then the keyboard language/mapping is prompted for when the system boots the first time if the system has a graphics console and keyboard attached.

_hp_root_disk

The `_hp_root_disk` string variable is set by the Ignite-UX GUI to contain the hardware path of the disk that the user has chosen to be the root disk. The value is initialized to the value of `_hp_primary_path` in `/opt/ignite/data/Rel_*/config`, and can be overridden in other `config` files. If no `config` file were to set the initial value, then it would default to the disk with the hardware path with the highest SCSI priority.

The standard installation configuration attempts to set the variable `_hp_root_disk` to the value of `_hp_primary_path` if `_hp_primary_path` is set. The variable `_hp_primary_path` holds the hardware path to the disk that the system is set up to boot from by default (set by `setboot` or selected at the BCH[41] prompt). If the primary path setup for the system points to a nonexistent device, `_hp_root_disk` is not be set by this configuration. Instead, Ignite-UX chooses what it determines to be the "best" device for the variable instead.

```
(_hp_primary_path != "" & !(_hp_root_disk ~ ".*"))
{
    init _hp_root_disk=_hp_primary_path
}
```

[40] Internal Terminal Emulator is an emulation of an HP style terminal in the kernel when a graphics display is the console device.
[41] Boot Console Handler: the firmware boot prompt on PA-RISC systems.

During an interactive installation or recovery, you can modify this value using the Ignite-UX GUI using the **Root Disk...** button on the **Basic** tab to get a selection of choices. It can also be changed using the **File system** tab in **Add/Remove Disks...** dialog box by changing the disks that are assigned to root volume or disk group. However, in the **Add/Remove Disks...** dialog box Ignite-UX chooses which of the disks (if more than one) is assigned to `_hp_root_disk` (initially anyway, after that they can be modified).

The standard configuration files use `_hp_root_disk visible_if false` to prevent this variable from being from visible or changeable using the **Additional** button in the Ignite-UX GUI.

_hp_boot_dev_path

The variable `_hp_boot_dev_path` is not defined by *instl_adm*(4). *Do not* rely on this description and the behavior defined here because they may change without notice.

If `bootsys` was used on a disk that was not destroyed during the installation, then the disk has an `AUTO` file left over from `bootsys`. Typically, you want to be able to boot from that disk after the installation is done. The disk that you booted from originally is pointed to by `_hp_boot_dev_path`.

The `bootsys` command stores the original `AUTO` file line as a commented-out string in the `AUTO` file itself (using '#' as the comment char). Ignite-UX uses the value in `_hp_boot_dev_path` to reset the `AUTO` file in the `LIF` of that device if that disk is not used in the install.

On Itanium®-based systems the `AUTO` file is stored in the Extensible Firmware Interface (EFI) partition. Ignite-UX treats that `AUTO` file exactly the same way as the PA-RISC `AUTO` file stored in a boot `LIF`.

Do not have this variable set by any custom configuration file; its use is intended only for internal use within Ignite-UX.

The standard configuration files use _hp_boot_dev_path visible_if false to prevent this variable being visible or changeable using the **Additional** button in the Ignite-UX GUI.

_hp_primary_path

The `_hp_primary_path` string variable is set during the initial startup of Ignite-UX on the client. The variable is set to the system's primary boot path (as set in stable storage and read with setboot). If the system's primary boot path is incorrectly set to a non-existent device, then `_hp_primary_path` is set to the empty string ("").

Use this variable for reference only, or to set other variables. Do not have a configuration file assign it a value. Ignite-UX sets the value.

_hp_primary_partition_size

The `_hp_primary_partition_size` variable controls the size of the HPUX partition on an EFI boot disk on HP Itanium®-based systems. Normally, the value of this variable is calculated as the remainder of the disk being installed to after the other partition sizes have been subtracted. For more information regarding the EFI and HPSP partitions, see "`_hp_efi_partition_size`" and "`_hp_service_partition_size`".

Typically, there is no reason to set the `_hp_primary_partition_size` variable in a custom configuration. If you do set this variable and the size is too large, it is automatically reduced; if set too low, part of the boot disk remains unused.

_hp_efi_partition_size

The _hp_efi_partition_size integer variable is set by the Ignite-UX GUI to indicate the size of the HP-UX Extensible Firmware Interface (EFI) boot partition on Itanium®-based systems. This disk partition holds loader software and other utilities used to boot the HP-UX operating system. This value defaults to a size that allows some space for additional boot partition content in future HP-UX releases. This variable is adjusted to the minimum EFI boot partition size if set to zero or a value less than the minimum. This variable is supported starting with HP-UX 11.23 and is only supported on Itanium®-based systems.

This variable sets the size of the EFI partition located on bootable disks on Itanium®-based systems. The minimum size is 150MB (default size 500MB) currently.

This variable can affect recoveries. For example, if there is no free space left on the boot disk and an Ignite-UX version is released that increases the size of the EFI partition, the recovery or installation requires manual intervention if the EFI partition size minimum is increased because of lack of disk space.

This variable can affect new installations or reinstallations. If you have sized the boot disk of a system to a certain size and have not included the size of the EFI Partition (and Service Partition, see "_hp_service_partition_size") you may not have enough disk space to install the system in the way that you want.

_hp_service_partition_size

The _hp_service_parition_size integer variable is set by the Ignite-UX GUI and indicates the size of the HP Service Partition on Itanium®-based systems. This disk partition holds diagnostics software, saved system and hardware state data, and other utilities used to verify and update hardware functionality. This value defaults to a size expected by the diagnostics software. A zero value indicates that no HP Service Partition is to be created. This variable is adjusted to the minimum HP Service Partition size if set to a non-zero value less than the minimum. This variable is supported starting with HP-UX 11.23 and only supported on Itanium®-based systems.

This variable defines the size of the HP Service Partition located on bootable disks on Itanium®-based systems. The minimum size is 400MB (default size 400MB) at this writing. If the size of the HP Service Partition is set to zero, the partition is not created.

This variable can affect recoveries. For example, if there is no free space left on the boot disk and an Ignite-UX version is released that increases the size of the HP Service Partition, the recovery or installation requires manual intervention if the HP Service Partition size minimum is increased because of lack of disk space

This variable can have an impact on new installations or reinstallations. If you have sized the boot disk of a system to a certain size and have not included the size of the HP Service Partition (and EFI Partition, see "_hp_efi_partition_size"), you may not have enough disk space to install the system in the way that you want.

_hp_root_grp_disks

The _hp_root_grp_disks integer variable indicates how many disks to put into the root volume group.

The itool wizard prompts you for the number of disks in the root volume/disk group and whether they should be striped. However, in the Ignite-UX GUI you must use the **Additional** button on the **Basic** tab to display the **# of disks in root VG**.

The default configuration described earlier (for B.11.11) sets _hp_root_grp_disks to an enum from one to the value of the internal variable num_disks. During an installation, Ignite-UX adds disks automatically to the root volume/disk group to make up this number. Because the disks are assigned automatically, you must ensure that the disks do not contain data that you do not want to lose. For example, in a SAN environment those disks may belong to another system.

_hp_root_grp_striped

The _hp_root_grp_striped string variable uses the possible values YES and NO to indicate if LVM data striping should be used on all disks in the root volume group if multiple disks are in the root group.

This variable is defined within the default installation configuration so you can easily specify whether the volumes within the root volume/disk group should be striped across all disks in the root volume/disk group.

The itool wizard prompts you to specify whether the disks in the root volume/disk groups should be striped (as well as to specify the number of disks in the root volume/disk group on the same screen). However, in the Ignite-UX GUI you must use the **Additional** button on the **Basic** tab to display the item **Stripe root VG disks?**.

Even though *instl_adm*(4) discusses LVM, striping also applies to VxVM (VxVM licensing may restrict its use). Remember that the root, boot, and primary swap+dump cannot be striped, so this only applies to other volumes in the root volume/disk group.

_hp_addnl_fs_free_pct

The _hp_addnl_fs_free_pct integer variable is used to control the amount of additional free space allocated to volumes beyond the space required to load the software. If this variable is not set, it defaults to 10 (10%).

The _hp_addnl_fs_free_pct variable can be changed using Additional free space % during an installation because of the following configuration.

```
_hp_addnl_fs_free_pct = { 0,1,2,3,4,5,6,7,8,9,10 }
init _hp_addnl_fs_free_pct = 10
_hp_addnl_fs_free_pct help_text "Additional free space %"
```

This variable is not created for recoveries.

The *instl_adm*(4) manpage states the following about how _hp_addnl_fs_free_pct affects the size calculations for volumes:

In addition to this minimum size, an additional percentage is added to the volumes to ensure sufficient room after the system is installed. This percentage is by default 10%, but can be controlled by the configuration file variable _hp_addnl_fs_free_pct. If the disk capacity is insufficient to accommodate this additional free space, this additional percentage value is continually reduced by 1/2 until a value is found that allows all the volumes to fit. If the volumes still do not fit with 0% additional space, then the installation is not allowed to proceed.

Therefore, the variable does nothing when there is no free disk space; however, if there is free disk space, volumes on the system may be expanded in an unexpected manner.

_hp_ignore_sw_impact

The _hp_ignore_sw_impact integer variable can be set to one if you want to disable all effects that the impacts statements declared in the software selections have on the volume size calculations. This may be helpful if you want to ensure that Ignite-UX does not automatically modify the file-system volume sizes.

Be careful when setting this variable to 1. Ignite-UX does not change the file system sizes based upon the impacts statements associated with software products that are being installed. The installer must correctly set file system sizes so the system does not fill file systems during installation.

So, when do you use this variable? To answer that you need to look at how impacts are created. Consider the following commands[42]:

```
$ print "f tmp/a 1024" | /opt/ignite/lbin/gen_impacts
    impacts = "tmp" 8Kb
$ print "f tmp/a 10240" | /opt/ignite/lbin/gen_impacts
    impacts = "tmp" 16Kb
```

The first command specifies a file size tmp/a of 1KB, but the impact is 8KB. The second command specifies a file size of tmp/a as 10KB, and the impact is 16KB.

Most systems these days use VxFS instead of HFS for file systems. VxFS file systems are extent-based file systems and can usually allocate (space allowing) blocks as small as 1KB to a file. When you have the impacts for a HFS file system applied to a VxFS file system, the size requirements can be significantly overstated because of the assumptions that gen_impacts[43] makes.

The gen_impacts command assumes that all files are located on HFS file systems and have an 8KB fragment size and a 64KB block size. The gen_impacts command gets the approximate size correct when the file is large enough to use indirect block to track the space used by the file.

These overstated impacts are not considered wrong, just conservative. Ignite-UX cannot predict when impacts are being generated or what kind of file system on which the software is installed. The correct approach is to take the conservative approach.

Lastly, consider the following example of what effect different fragment sizes have on the impacts generated for a depot containing a copy of the B.11.11 Version Mission Critical (MC) Operating Environment (OE) with the default fragment size (8KB), a 2KB fragment size, and a 1KB fragment size:

```
# /usr/sbin/swlist -l file -d -a type -a name -a size @/depot/11iv1_mc/core |
awk ' $3 == "f" || $3 == "d" { printf("%s %s %s\n", $3, $2, $4) } ' |
/opt/ignite/lbin/gen_impacts -l 2 -f 1024
    impacts = "/" 1138Kb
    impacts = "/usr/lib" 599090Kb
```

---

[42] You do not call gen_impacts in this way, because the format that it accepts is not documented. This example is only to illustrate how impacts are generated.

[43] The archive_impact and make_arch_config commands both call gen_impacts to generate impacts statements.

```
    impacts = "/sbin/lib" 561Kb
    impacts = "/usr/conf" 107843Kb
    impacts = "/sbin/init.d" 664Kb
…

# /usr/sbin/swlist -l file -d -a type -a name -a size @/ depot/11iv1_mc/core|
awk ' $3 == "f" || $3 == "d" { printf("%s %s %s\n", $3, $2, $4) } ' |
/opt/ignite/lbin/gen_impacts -l 2 -f 2048
    impacts = "/" 1138Kb
    impacts = "/usr/lib" 603124Kb
    impacts = "/sbin/lib" 578Kb
    impacts = "/usr/conf" 108359Kb
    impacts = "/sbin/init.d" 734Kb
…

# /usr/sbin/swlist -l file -d -a type -a name -a size @/depot/11iv1_mc/core |
awk ' $3 == "f" || $3 == "d" { printf("%s %s %s\n", $3, $2, $4) } ' |
/opt/ignite/lbin/gen_impacts -l 2 -f 8192
    impacts = "/" 1138Kb
    impacts = "/usr/lib" 632140Kb
    impacts = "/sbin/lib" 684Kb
    impacts = "/usr/conf" 111703Kb
    impacts = "/sbin/init.d" 1256Kb
…
```

The size differences in the first four impacts are summarized in Table 4 (the percentage difference is between the 2KB and 8KB fragments compared to the 1KB fragment impacts):

Table 4

Impacts	Size (1KB)	Size(2KB)	% difference	Size (8KB)	%difference
/	1138	1138	0.0%	1138	0.0%
/usr/lib	599090	603124	0.7%	632140	5.5%
/sbin/lib	561	578	3.0%	684	21.9%
/usr/conf	107843	108359	0.5%	111703	3.6%
/sbin/init.d	664	743	11.9%	1256	89.2%

The percentage increases are somewhat variable but when more space is used for the larger impacts, the larger the fragment size. The impacts are made much larger when a directory structure contains many very small files.

The reason why "/" is the same size is that `gen_impacts` allocates 1KB to an impact level whenever it sees a directory mentioned. The "/" directory in this case appears 1138 times in the `swlist` output (so multiple mentions of a directory in the output from `swlist` can also bloat impacts).

Caution:
If you change or somehow recalculate impacts in any way, you *must* be very careful. Impacts that do not fully take into account the software impacts to a system may lead to configurations that cannot install a system (because of file system full problems).

_hp_custom_sys

The `_hp_custom_sys` string variable is used in conditional statements surrounding network and system identity information when the Ignite-UX GUI writes out a configuration file. This variable can be used in conditionals in configuration files to define multiple sets of network parameters that can be selected from the **Additional** screen. You can see how this variable is used by clicking the **Save As** button during an installation after modifying the parameters under the **System** tab and looking at the resulting file in the `/var/opt/ignite/saved_cfgs` directory.

Never use the `_hp_custom_sys` variable in custom configurations. Ignite-UX uses this variable in the configuration files it can produce so setting it's value outside of those files may interfere with it's use by Ignite-UX.

_hp_lanadmin_args

If your network requires that the default MTU or speed be set using the `lanadmin` command to operate correctly, you can specify the arguments to the lanadmin command using the `_hp_lanadmin_args` variable. This variable setting must be set in the `INSTALLFS` file using the `instl_adm` command. Any change made with this variable only affects the installation and is not permanently applied to the system. For example, to set the MTU size to 1500, the line would be:

```
init _hp_lanadmin_args="-M 1500"
```

To set the speed for a 100-Base-T interface to full duplex (the default is half-duplex) you could use the setting:

```
init _hp_lanadmin_args="-S 1"
```

Setting the MTU value with `lanadmin` only works for the NIO/HPPB FDDI interface.

Additional `lanadmin` libraries have been added over time to allow many interfaces to be controlled with the `-X` option to `lanadmin`.

This variable is only useful when placed into the installation file system so the changes are available at boot time. For example:

```
( lan[].driver ~ "btlan" ) {
    _hp_lanadmin_args="-X 100FD"
```

```
}
```

You might choose to place the preceding configuration lines into the installation file system if you have systems (using LAN interfaces controlled by the `btlan` driver[44]) connected to switches that have been set to operate only at 100 Full Duplex.

The reason you would need to do this is that there are no startup scripts when installing a system. The `btlan` driver attempts to auto-negotiate the speed and duplex settings with the device to which it is connected. If the switch is set to 100 Full Duplex the auto-negotiation fails and the system runs at 100 Half Duplex.

When the system and the switch attempt to communicate using mismatched duplexes, the values result in problems and performance suffers. This leads to either very long installation times or a failed installation.

For additional information, see "Problems that can be solved with **_hp_lanadmin_args**".

_hp_nfs_mount_opts

The `_hp_nfs_mount_opts` string variable is set in the `INSTALLFS` file and is used to supply additional options to the NFS mounts that are performed during the installation. This is intended for use when the default options are not appropriate for your network. (Refer to *`mount_nfs`*`(1M)` for valid options). For example, to set the read and write NFS buffer size to 1K:

```
init _hp_nfs_mount_opts="-orsize=1024 -owsize=1024"
```

This option must be given in the installation file system for it to take effect, although a read buffer size of 4-8KB performs better for recovery over a local network. Otherwise, if you were to read the value of this variable from the Ignite-UX server, the NFS file system would already be mounted and the options would be too late to be applied.

Before Ignite-UX version C.6.0.*x* the *`instl_adm`*`(4)` manpage is incorrect; the NFS mount command accepts one and only one `-o` option. All of the options to the `-o` parameter must be given in a comma separated list. The following example does work:

```
init _hp_nfs_mount_opts="-orsize=1024,wsize=1024"
```

Do not use NFS mount options documented in *`mount_nfs`*`(1M)` that interfere with the correct operation of Ignite-UX such as the following:

`-oro` — If you mount the file system read-only it is very hard for Ignite-UX to update files on it. It is also incompatible with the `-orw` option provided by Ignite-UX (some file systems are mounted read-only by Ignite-UX so the `-orw` option should not be given either).

`-obg` — Never place the mount attempts into the background as once the mount command returns successfully, Ignite-UX assumes that the file system has mounted and

[44] The installation kernel used to install and recover HP-UX 11.0 and HP-UX 11i v1 (B.11.11) systems is based upon the HP-UX 11i v1 kernel. When referring to kernel driver names you *must* use the driver name from HP-UX 11i v1.

then continues. Ignite-UX provides a retry mechanism using the `_hp_nfs_mount_retries` variable.

`-overs=2` — Unless absolutely necessary do not use this option as NFS v2 does not allow you to use large files (golden image and recovery archives more than 2GB in size fails to install/recover).

Other NFS options may be used as needed unless they interfere with the operation of Ignite-UX.

_hp_nfs_mount_retries

The `_hp_nfs_mount_retries` integer variable is used to modify the default number of times that the NFS mounts are retried before failing. If this variable is not set, the mounts are retried 4 times before giving up. If you need to change this default, this variable should be specified in the `INSTALLFS` file. For example, to set the number of retries to 8:

```
init _hp_nfs_mount_retries=8
```

This is the number of retry attempts that Ignite-UX performs when attempting to mount an NFS file system. If Ignite-UX attempts to retry an NFS mount, messages similar to the following appear on the console and in the client's `install.log` file:

```
NOTE:    Retrying: "/usr/sbin/mount -orw
10.0.0.1:/var/opt/ignite/clients /var/opt/ignite/clients"
NOTE:    Retrying: "/usr/sbin/mount -orw
10.0.0.1:/var/opt/ignite/clients /var/opt/ignite/clients"
NOTE:    Retrying: "/usr/sbin/mount -orw
10.0.0.1:/var/opt/ignite/clients /var/opt/ignite/clients"
...
```

The messages continue until either the amount of retries given in `_hp_nfs_mount_retries` is exceeded or the NFS file system mounts successfully.

_hp_tftp_cmds

The `_hp_tftp_cmds` string variable can be specified in the `INSTALLFS` file to supply additional instructions to the `tftp` commands that are used to transfer data during an installation. The commands supplied with this variable are passed as input to the `tftp` command along with the usual commands supplied by Ignite-UX. The most likely use of this would be to modify the retransmission-timeout (`rexmt`) and overall timeout (`timeout`) values. The default values that Ignite-UX uses are `rexmt=2 timeout=25`. (Refer to *`tftp`*`(1)` for more details). The string assigned to this variable should contain one `tftp` command statement per line. For example:

```
init _hp_tftp_cmds="rexmt 5
```

```
timeout 40"
```

Setting this variable can increase the number of retransmissions and the timeout. You may want to increase these values if the system being installed is a long way (network-wise) from the Ignite-UX server or if you have a fairly unreliable or overloaded network.

_hp_hide_other_disks

The `_hp_hide_other_disks` string variable can be set to one or more space-separated hardware paths of disks that should be "hidden" from being configured or otherwise modified during the install. It allows more disks to be hidden than what is provided by the `hide_boot_disk` keyword.

This variable is not especially useful for installation purposes, as you need to know all of the disk hardware paths that need to be hidden, which is difficult at installation time unless you know exactly what is in the system beforehand and what it looks like.

This is more useful during recoveries as Ignite-UX currently does them, as you can ensure that some disks cannot be accessed or modified during a recovery session.

_hp_saved_detail_level

The `_hp_save_detail_level` internal string variable is used in configuration files that have been created by a **Save-As** operation in the Ignite-UX GUI. Like `_hp_cfg_detail_level`, it contains a list of option characters that represent which aspects of the configuration file have been modified. The format is identical to the `_hp_cfg_detail_level` variable.

You should not use this variable in custom configurations. If you are using `itool` or other Ignite-UX internal commands to create configuration files that place this variable into a configuration file, you should remove any reference to this variable.

_hp_os_bitness

Beginning at HP-UX 11.00, the `_hp_os_bitness` string variable is used to package software for running on a 32-bit operating system, 64-bit operating system, or both. The `_hp_os_bitness` variable is set to either `32` or `64` when an operating system is chosen. This is normally done in a configuration file by keying off the `sw_sel` for a 32- or 64-bit operating system. The `sw_sel` statements for certain applications rely on the setting of this variable to tell which version of the application to install.

This variable is set by the core operating system `sw_sel` that was selected, for example:

```
sw_sel "HPUXBase64" {
    description = "HP-UX 64-bit Base OS"
...
}
(sw_sel "HPUXBase64") {
    _hp_os_bitness = "64"
}
```

When the software HPUXBase64 is selected (so the test "(sw_sel "HPUXBase64")" returns true) the variable _hp_os_bitness is set to 64. The make_config command automatically does this for you. However, if you manually write configuration files containing definitions of a core operating system depot or archive, you must set this variable to the correct bitness.

_hp_force_autoboot

The _hp_force_autoboot string variable is used to modify the behavior of Ignite-UX with respect to stable store's autoboot flag. The Ignite-UX configuration process has two parts separated by a reboot. By default (_hp_force_autoboot="YES"), Ignite-UX guarantees that autoboot is set during the installation process and then reset to its previous state (if necessary) at the end of the installation. If _hp_force_autoboot="NO", Ignite-UX does not modify the autoboot flag in stable storage. This may mean that you do not have to do a manual boot from the primary path between the two parts of the Ignite-UX installation.

Ignite-UX normally forces the system to perform the first reboot automatically when rebooting from the installation kernel to the newly created or recovered kernel. This reboot is forced no matter what the autoboot flag is set to (refer to *setboot*(1M)). Ignite-UX forces it on if it was off; prior to the final reboot Ignite-UX returns the autoboot flag to off. If _hp_force_autoboot is set to NO and the system's autoboot flag is set to off, the system cannot complete the recovery and boot the final recovered or installed system, which leaves the system at the BCH, prompt).

Important:
If you disable autoboot and then install a system, the system never executes a final reboot without manual intervention. HP does not support booting into any other mode than the default multi-user run level at this point. The installation or recovery session has not yet completed and some steps must still be performed by Ignite-UX. Booting into anything other than the default run level does not allow the installation or recovery to complete. This is the intended behavior of Ignite-UX and not a flaw. In general (except when investigating Ignite-UX issues on advice from HP Support), you should never change the value of the _hp_force_autoboot variable.

_hp_ikernel_os_release

The _hp_ikernel_os_release variable is not defined in the Ignite-UX documentation and it should never be set explicitly by any custom written Ignite-UX configuration files.

When a client boots, part of starting Ignite-UX for installation or recovery is to create a file called "host.info". As of Ignite-UX version C.6.0.*x*, this variable is added to the host.info file and gives the HP-UX revision that the installation kernel is running. Ignite-UX uses this information to prevent you from selecting HP-UX releases that cannot be installed by the installation kernel.

For example, with Ignite-UX version C.6.0.*x* for PA-RISC systems, the B.11.11 installation kernel is used to install B.11.00 and B.11.11 client systems but cannot be used to install B.11.23 systems. Therefore, if _hp_ikernel_os_release is set to B.11.11 Ignite-UX will not allow you to install HP-UX B.11.23.

_hp_current_client_release

The _hp_current_client_release variable determines what release of HP-UX the client is currently running and is used by the ignite and make_net_recovery commands. This variable should not be set in any custom-written configuration file. It is not used during installation or recovery.

_HP_CLONING

The _HP_CLONING variable is only created by make_net_recovery and make_tape_recovery as part of the configuration for a recovery archive. There seems to be some confusion about exactly what this variable does. It is defined as follows in the control_cfg file:

```
enum _HP_CLONING
_HP_CLONING help_text "Cloning to different HW?"
_HP_CLONING = { "TRUE", "FALSE" }
(MODEL == "9000/800/SD16000")
{ init _HP_CLONING = "FALSE" }
else
{ init _HP_CLONING = "TRUE" }
```

This defines _HP_CLONING as an enum and means that it can only have a value from one of the list of possible values that are defined for it (on the third line of the definition). The second line defines the help text that appears when using the **Additional** button on the **Basic** tab. If the model string matches the pattern, it defaults to FALSE; otherwise, it defaults to TRUE.

If you are using a recovery archive containing this configuration to clone one system to another with the same model string, you should manually change this value to TRUE using the **Additional** button on the **Basic** tab. This is to force a new kernel to be created. Although the model of system may be the same, it may have different I/O cards present. If the driver for those I/O cards is not present in the kernel being recovered, the final system will not have the drivers and those I/O cards will not be claimed.

The code that does this is in one of the scripts provided with Ignite-UX that is run during a recovery (os_arch_post_1):

```
# If cloning to a different model of machine, _HP_CLONING will
# be true (it can also be set via the Additional UI screen).  In
# this case remove the kernel which will allow IUX to rebuild it,
# plus remove device files which will get rebuilt by insf or
# might be inappropriate.
if [[ "$_HP_CLONING" = "TRUE" ]] ; then
    remove_devs
    rm -f /stand/vmunix
```

```
fi
```

This variable does not control anything else.

_hp_console_verbosity

The `_hp_console_verbosity` variable allows you to reduce the amount of information that is printed to the screen when Ignite-UX executes a recovery or an installation. It has a default value of five that sets Ignite-UX to print everything; however, it only has to be three or more to print everything.

When set to two or less, it greatly reduces the amount of messages printed on the console. The messages not printed on the console are printed to the `install.log` file. This variable maybe useful when creating media for use by non-technical personnel who may not understand the messages being printed or may become concerned with some of the warning messages that Ignite-UX can print that may not be indicative of a real problem.

This variable is set in the installation file system. The following example reduces the number of messages written to the console during a recovery or installation by setting the variable to zero:

```
init _hp_console_verbosity=0
```

Warning:
If you reduce the amount of messages printed and a failure occurs, this may cause difficulties when attempting to determine the root cause of failures that occur early in an installation or recovery when the `install.log` file has not yet been moved to permanent storage.

_hp_patch_save_files

The `_hp_patch_save_files` variable is not actually a special variable and by itself, it means nothing to Ignite-UX. It is a construct used within the `hw_patches_cfg` file in the per release data directory (`/opt/ignite/data/Rel_B.##.##`). This construct is an `enum` that is accessible from the **Additional** button on the **Basic** tab in the Ignite-UX GUI. The following is an example of the `_hp_patch_save_files enum` definition:

```
#
# _hp_patch_save_files is used to determine if patches loaded from the
# core are committed, or not committed and can later be backed out.
#
enum _hp_patch_save_files
_hp_patch_save_files = { "YES", "NO" }
init _hp_patch_save_files = "YES"
_hp_patch_save_files help_text "Save patched files?"
```

However, the description in the previous example is wrong because the patches are not committed. This variable construct only controls whether the `swinstall option -x patch_save_files=true or false` command is added to the `swinstall` command line.

Later in the same file, the following construct is found:

```
(_hp_patch_save_files == "NO")
{
  sd_command_line += " -xpatch_save_files=false "
}
else
{
  # This is actually the default...
  sd_command_line += " -xpatch_save_files=true "
}
```

Again, this variable construct is specific to *one* configuration file. It is not a special variable that controls all software installations during an Ignite-UX installation session.

_hp_umask

The `_hp_umask` variable allows you to set the `umask` to a custom value during a recovery or installation session. If not set, the `umask` value used by Ignite-UX defaults to 022 (removes write permission from other and group).

When a recovery or installation session first starts the `umask` value is set to the value 022; this is not configurable. The `umask` defined by `_hp_umask` is only set once the configuration user interface has finished and phase one of the recovery or installation session starts.

After the first reboot and the recovery or installation continues, the `umask` value from `_hp_umask` is used to set the `umask` again.

Warning:
HP does not test Ignite-UX with any `umask` setting other than 022; setting the `umask` to a more restrictive value may cause installation failures. HP does not consider installation failures caused by setting `_hp_umask` to a value other than the default to be a defect in Ignite-UX.

Configuration for software to be installed

This section discusses the configuration files required for software, beginning with applications.

Application software depots

The following example system has multiple depots, one for the HP-UX 11i v1 (B.11.11) core operating system depot and the other two for different versions of vPars (HP product number T1335AC):

```
# swlist -l depot
# Initializing...
# Target "box2 " has the following depot(s):
  /var/opt/ignite/depots/Rel_B.11.11/core
  /var/opt/ignite/depots/T1335AC_A.02.02.00
  /var/opt/ignite/depots/recovery_cmds
  /var/opt/ignite/depots/T1335AC_A.02.03.02
```

You must create some configuration files for the depots with the `make_config` command:

```
# make_config -l 2 -s /var/opt/ignite/depots/T1335AC_A.02.03.02 \
  -c /var/opt/ignite/data/Rel_B.11.11/vpars_A.02.03.02.cfg
NOTE:    make_config can sometimes take a long time to complete. Please be
         patient!
# make_config -l 2 -s /var/opt/ignite/depots/T1335AC_A.02.03.02 \
  -c /var/opt/ignite/data/Rel_B.11.11/vpars_A.02.02.00.cfg
NOTE:    make_config can sometimes take a long time to complete. Please be
         patient!
```

The configuration file produced for version A.02.02.00 of vPars contains the `sw_source` statement for the depot. The depot is available over the network, it is an SD depot, and is located on the server 10.0.0.1 at `/var/opt/ignite/depots/T1335AC_A.02.02.00`:

```
# cat vpars_A.02.02.00.cfg
#############################################################
##  Software Sources
#############################################################

sw_source "/var/opt/ignite/depots/T1335AC_A.02.02.00" {
    source_type = "NET"
    sd_server = "10.0.0.1"
    sd_depot_dir = "/var/opt/ignite/depots/T1335AC_A.02.02.00"
    source_format = SD
```

```
}
```

Next is patch software. The patch is only applicable to 64-bit systems (so it becomes invisible on 32-bit systems):

```
#########################################################
##  Other Software
#########################################################

sw_sel "b_PHSS_28764" {
    (_hp_os_bitness == "64") {
          description = "(PHSS_28764) vPar (A.02.02) monitor cumulative patch"
          sw_source = "/var/opt/ignite/depots/T1335AC_A.02.02.00"
          sw_category = "Uncategorized"
          sd_software_list = "b_PHSS_28764,r=1.0,a=HP-UX_B.11.11_64,v=HP"
          impacts = "/stand" 2288Kb
          visible_if = TRUE
    }
    else {
          visible_if = FALSE
    }
}
```

Although not shown here, `make_bundles` was run to create the bundle `b_PHSS_28764`.

Next is the vPars product:

```
sw_sel "T1335AC" {
    (_hp_os_bitness == "64") {
          description = "HP-UX Virtual Partitions"
          sw_source = "/var/opt/ignite/depots/T1335AC_A.02.02.00"
          sw_category = "OrderedApps"
          sd_software_list = "T1335AC,r=A.02.02.00,a=HP-UX_B.11.11_64,v=HP"
          impacts = "/usr" 2Kb
          impacts = "/etc/rc.config.d" 24Kb
          impacts = "/stand" 1136Kb
          impacts = "/usr/share" 1640Kb
          impacts = "/usr/sbin" 3712Kb
          impacts = "/usr/lib" 2747Kb
          impacts = "/usr/lbin" 1376Kb
```

```
            impacts = "/usr/contrib" 15304Kb
            impacts = "/sbin" 1595Kb
            impacts = "/usr/conf" 9578Kb
            impacts = "/usr/include" 640Kb
            impacts = "/usr/newconfig" 5664Kb
            impacts = "/usr/ccs" 10248Kb
            impacts = "/usr/bin" 208Kb
            impacts = "/sbin/init.d" 40Kb
            impacts = "/" 2Kb
            visible_if = TRUE
      }
      else {
            visible_if = FALSE
      }
}
```

The configuration file is not yet 100% complete and requires a manual change that causes problems. Specifically, the patch defined by the `sw_sel`, `b_PHSS_28764` needs to be installed at the same time as `T1335AC`. Therefore, you need to make a small change to the `sw_sel` clause for `T1335AC`:

```
sw_sel "T1335AC" {
    (_hp_os_bitness == "64") {
          description = "HP-UX Virtual Partitions"
          sw_source = "/var/opt/ignite/depots/T1335AC_A.02.02.00"
          sw_category = "OrderedApps"
          sd_software_list = "T1335AC,r=A.02.02.00,a=HP-UX_B.11.11_64,v=HP"
          impacts = "/usr" 2Kb
          impacts = "/etc/rc.config.d" 24Kb
          impacts = "/stand" 1136Kb
          impacts = "/usr/share" 1640Kb
          impacts = "/usr/sbin" 3712Kb
          impacts = "/usr/lib" 2747Kb
          impacts = "/usr/lbin" 1376Kb
          impacts = "/usr/contrib" 15304Kb
          impacts = "/sbin" 1595Kb
          impacts = "/usr/conf" 9578Kb
          impacts = "/usr/include" 640Kb
          impacts = "/usr/newconfig" 5664Kb
          impacts = "/usr/ccs" 10248Kb
          impacts = "/usr/bin" 208Kb
```

```
            impacts = "/sbin/init.d" 40Kb
            impacts = "/" 2Kb
            visible_if = TRUE
            corequisite="b_PHSS_28764"    ←
      }
      else {
            visible_if = FALSE
      }
}
```

Because you want the patch to be installed automatically with vPars, you need to manually add the `corequisite` configuration line. When located in the same depot, the default `swinstall` behavior of `autoselect_patches=true` automatically selects the patch with the installation of the relevant vPars bundle. The configuration file changes used here are strictly needed only when the patch is not in the same depot.

Now you can look at the vPars software configuration file for the other version of vPars (version A.02.03.02). The `sw_source` definition is to the previous version except the depot is in a different location, so the `sw_source` has a different name as well.

```
# cat vpars_A.02.03.02.cfg
#############################################################
##  Software Sources
#############################################################

sw_source "/var/opt/ignite/depots/T1335AC_A.02.03.02" {
    source_type = "NET"
    sd_server = "10.0.0.1"
    sd_depot_dir = "/var/opt/ignite/depots/T1335AC_A.02.03.02"
    source_format = SD
}
```

Now vPars version A.02.03.02 contains many software products, as depicted in the following:

```
#########################################################
##  Other Software
#########################################################

sw_sel "VPARMGR" {
    description = "Virtual Partition Manager - HP-UX"
    sw_source = "/var/opt/ignite/depots/T1335AC_A.02.03.02"
    sw_category = "OrderedApps"
```

```
    sd_software_list = "VPARMGR,r=B.11.11.01.02,a=HP-UX_B.11.11_32/64,v=HP"
    impacts = "/var" 1Kb
    impacts = "/opt/webadmin" 1544Kb
    impacts = "/opt/vparmgr" 204Kb
    impacts = "/opt" 1Kb
    impacts = "/usr/share" 8Kb
}


sw_sel "VPARDoc" {
    description = "VPAR Documentation Bundle"
    sw_source = "/var/opt/ignite/depots/T1335AC_A.02.03.02"
    sw_category = "HPUXAdditions"
    sd_software_list = "VPARDoc,r=B.11.11.59,a=HP-UX_B.11.11_32/64,v=HP"
    impacts = "/tmp" 8Kb
    impacts = "/cdrom/vParsWINSTALL" 49664Kb
    impacts = "/cdrom/DOCS" 1192Kb
}

sw_sel "T1335AC" {
    (_hp_os_bitness == "64") {
         description = "HP-UX Virtual Partitions"
         sw_source = "/var/opt/ignite/depots/T1335AC_A.02.03.02"
         sw_category = "OrderedApps"
         sd_software_list = "T1335AC,r=A.02.03.02,a=HP-UX_B.11.11_64,v=HP"
         impacts = "/usr" 2Kb
         impacts = "/etc/rc.config.d" 24Kb
         impacts = "/stand" 1176Kb
         impacts = "/usr/share" 1640Kb
         impacts = "/usr/sbin" 3712Kb
         impacts = "/usr/lib" 2747Kb
         impacts = "/usr/lbin" 1376Kb
         impacts = "/usr/contrib" 15304Kb
         impacts = "/sbin" 1603Kb
         impacts = "/usr/conf" 9194Kb
         impacts = "/usr/include" 640Kb
         impacts = "/usr/newconfig" 5664Kb
         impacts = "/usr/ccs" 10248Kb
         impacts = "/usr/bin" 208Kb
         impacts = "/sbin/init.d" 40Kb
```

```
            impacts = "/" 2Kb
            visible_if = TRUE
        }
        else {
            visible_if = FALSE
        }
    }


    sw_sel "KRMonitor" {
        description = "EMS Kernel Resource Monitor"
        sw_source = "/var/opt/ignite/depots/T1335AC_A.02.03.02"
        sw_category = "HPUXAdditions"
        sd_software_list = "KRMonitor,r=B.11.11.04,a=HP-UX_B.11.11_32/64,v=HP"
    ...
```

Using both of these configuration files together (or the configuration file `/var/opt/ignite/data/Rel_B.11.11/vpars_A.02.03.02.cfg` with anything else) causes conflicts unless manual changes are made.

When `make_config` finds two or more bundles of the same name in one depot it places a combination of the `revision`, `architecture`, and `vendor` into the `sw_sel` clause name (enough of one or more of them to make the `sw_sel` name unique). When `make_config` finds that a bundle name is unique in the depot, it does not place the `revision`, `architecture`, or `vendor` in the name of the `sw_sel` clause.

If you included both of the preceding configurations, the duplicate names (in this case "`T1335AC`") would cause issues. The information in the first definition of the `sw_sel` is overruled with the information from the second definition. The reason for this is that `sw_sel` clauses are stored by name, `T1335AC`, without regard to the configuration file from which they were derived.

To prevent this issue, you need to distinguish between the products in the first configuration file and the second configuration file. So change vPars A.02.02 to the following:

```
    sw_sel "T1335AC,r=A.02.02.00" {
        (_hp_os_bitness == "64") {
    ...
            exrequisite="T1335AC,r=A.02.03.02"
            visible_if = TRUE
        }
        else {
            visible_if = FALSE
        }
    }
```

This puts the `revision number` into the name of the `sw_sel`. When you have multiple versions of a product available to an installation session (but potentially in different depots), it is a good idea to manually include the `version number` in the configuration. In the second configuration file the same product needs to become the following:

```
sw_sel "T1335AC,r=A.02.03.02" {
    (_hp_os_bitness == "64") {
...

          exrequisite="T1335AC,r=A.02.02.00"
          visible_if = TRUE
    }
    else {
          visible_if = FALSE
    }
}
```

The `exrequisite` statements are added to make sure that you cannot select both products at the same time. In this case, to change from one revision of the vPars bundle to the other, simply select it. The originally selected version is automatically unselected because of the `exrequisite` statement.

Core operating system depot configuration

The following example shows a configuration file for a core operating system depot:

```
# cat core.cfg
############################################################
##  Software Sources
############################################################

sw_source "core" {
    description = "HP-UX Core Software"
    source_format = SD
    sd_server = "10.0.0.1"
    sd_depot_dir = "/var/opt/ignite/depots/Rel_B.11.11/core"
    source_type = "NET"
    load_order = 0
}
```

The `sw_source` definition for the core operating system depot contains the information needed to define an SD depot. The `load_order` of 0 (zero) is important for a core operating system depot. The load order determines the order in which Ignite-UX loads

the software. Other software products can be loaded at load order 0 except when the core operating system is an archive rather than a depot. When loading a core operating system archive, you can have one and only one `sw_sel` installed at `load_order` 0.

If you place any other software in the core operating system depot, `swinstall` resolves dependencies from the source depot that it is installing from before attempting to satisfy dependencies from the target. If you were to use "`-x reinstall=true`" on a `swinstall` command to install a product that had dependencies on the core operating system, you could also reinstall parts of the core operating system along with the product. The easiest way to prevent this from happening is to avoid placing additional software products in the core operating system depot or to never load software from that depot outside of Ignite-UX (you may know to never use `-x reinstall=true` but you may want to use that depot in the future with that option).

Next are the `sw_sel` definitions of the 32-bit and 64-bit Base HP-UX bundles. Effectively the same definitions are generated for a core operating system depot. Never include the definitions of multiple core operating system depots within one clause in the `/var/opt/ignite/INDEX` file because they start to override each other and cause issues with the install.

In the preceding example, the impacts statements for the bundles have been removed to make the clauses easier to read. Now, look at the 64-bit `sw_sel` clause. The name of the `sw_sel` clause is `HPUXBase64`. Its description is set to `HP-UX 64bit Base OS`. It has a `sw_source` set to `core` that was defined earlier. Always try to place the `sw_source` definition and the software products that reference it in the same configuration file to prevent dependencies between configuration files.

Next is the software category. For more information on categories and other software attributes (including special handling for some attributes), see the section on "Categories and other Ignite-UX software attributes" because some categories are handled specially. Some categories are not shown to the user as being available for selection.

The `sd_software_list` next defines the software that this `sw_sel` references. In this case, it is the bundle HPUXBase64 with the attributes `revision` ("r=") B.11.11, `architecture` ("a=") HP-UX_B.11.11, and `vendor` ("v=") HP.

Next `exrequisite = sw_category` makes sure that no other software with the same software category can be selected at the same time as this bundle. This automatically unselects any other `sw_sel` with the same software category set. This is to prevent both the 64- and 32-bit bundles from being selected simultaneously. That is the end of the software definition.

The following example tests to see if `sw_sel HPUXBase64` is selected (the surrounding parenthesis make this a test) and the variable `_hp_os_bitness` is set to 64, if the 64-bit Base operating system has been selected.

```
#########################################################
##  HPUX Base OS
#########################################################

sw_sel "HPUXBase64" {
```

```
    description = "HP-UX 64-bit Base OS"
    sw_source = "core"
    sw_category = "HPUXBaseOS"
    sd_software_list = "HPUXBase64,r=B.11.11,a=HP-UX_B.11.11_64,v=HP"
...
    exrequisite = sw_category
}
(sw_sel "HPUXBase64") {
    _hp_os_bitness = "64"
}
```

The following example is the definition of the 32-bit Base HP-UX bundle. It is very similar to the 64-bit one. It also has the `exrequisite = sw_category` attribute which prevents the selection of the 32-bit and 64-bit Base HP-UX bundle at the same time.

```
sw_sel "HPUXBase32" {
    description = "HP-UX 32-bit Base OS"
    sw_source = "core"
    sw_category = "HPUXBaseOS"
    sd_software_list = "HPUXBase32,r=B.11.11,a=HP-UX_B.11.11_32,v=HP"
...
    exrequisite = sw_category
}
(sw_sel "HPUXBase32") {
    _hp_os_bitness = "32"
}
```

Next are the operating environment (OE) definitions for the Base HP-UX 11i v1 OEs. These definitions determine whether the system can run a 64-bit operating system or not. Ignoring the contents of the `sw_sel` clauses, the `sw_sel "OE90BaseOS64"` is automatically selected if the system can run 64-bit HP-UX. The `visible_if` prevents the Operating Environment (OE) from being seen if the system cannot run a 64-bit version of HP-UX).

```
init sw_sel "OE90BaseOS64" {
    description = "HP-UX 11i Base OS-64bit"
    sw_source = "core"
    sw_category = "HPUXEnvironments"
    corequisite = "HPUXBase64"
    visible_if = can_run_64bit
} = (can_run_64bit)
```

The 32-bit OE is only visible if the system can run a 32-bit version of HP-UX; remember some older systems can run either. The 32-bit OE is automatically selected if the system

cannot run 64-bit HP-UX or if the system can run 32-bit HP-UX, and the 64-bit Base OE is not currently selected (so you must end up with one or the other selected).

```
init sw_sel "OE91BaseOS32" {
    description = "HP-UX 11i Base OS-32bit"
    sw_source = "core"
    sw_category = "HPUXEnvironments"
    corequisite = "HPUXBase32"
    visible_if = can_run_32bit
} = (! can_run_64bit) | (can_run_32bit) & !(sw_sel "OE90BaseOS64")
```

Now you have some code selecting on the model that it runs on and choosing a language (the `sw_sel` clause for `"English"` is not shown in this section but it does exist in the configuration file).

The items of most interest here are the regular expressions shown since they need improvement. The next section discusses what the regular expressions do.

The regular expression `"9000/8.*"` means match for the pattern `9000/8` followed by any zero or more of any character. The `"."` means any character and the following `"*"` changes that to mean zero (0) or more of that character. The regular expression `"9000/8"` is more precise since it just matches any string containing `"9000/8"`; the `".*"` is superfluous. Since the expression is not anchored,[45] nothing else is needed in the regular expression.

The same thing applies to the `"ia64 .* server .*"`, it could be more simply written as `"ia64.*server"`. Since it is an unanchored expression, you do not have to worry about what comes before or after the string. The changed expression matches ia64 followed by zero or more of any character followed by server. Again, because it is unanchored you do not care about what comes after the server part of the expression so there is no need for an explicit `".*"` at the end of the expression.

For PA-RISC servers and Itanium®-based servers, when the Base HP-UX OE is selected, the `sw_sel "English"` is automatically selected.

```
HARDWARE_MODEL ~ "9000/8.*" | MODEL ~ "ia64 .* server .*" {

    (sw_sel "OE91BaseOS32") | (sw_sel "OE90BaseOS64") {
        init sw_sel "English" = TRUE
    }
```

[45] Regular Expression anchoring involves a `"^"` to indicate the start of the line to match against, and `"$"` to indicate the end of the line. For example `'print "baaaa" |grep "^aaaa"'` does not match anything because the text being passed through grep starts with a `"b"`. The `"^aaaa"` says to only match lines starting with `"aaaa"` at the very start of the line, so the expression is anchored to the start of the line. The use of `"$"` similarly anchors the expression to the end of the line.

```
} # end of hardware-specific section


else {
```

The same thing discussed previously about regular expressions also applies to the workstation regular expressions that follow.

If you have a workstation, initialize the `sw_sel clause, "Global",` to be `TRUE`.

```
HARDWARE_MODEL ~ "9000/7.*" | MODEL ~ "ia64 .* workstation .*" {


    (sw_sel "OE91BaseOS32") | (sw_sel "OE90BaseOS64") {
        init sw_sel "Global" = TRUE
    }
} # end of hardware-specific section
```

Otherwise the `sw_sel "English"` is set to `TRUE`.

```
else {


    (sw_sel "OE91BaseOS32") | (sw_sel "OE90BaseOS64") {
        init sw_sel "English" = TRUE
    }
}
}
```

The following example shows the OE definitions for all of the OEs on the B.11.11 OE media. The definitions are similar to the other `sw_sel` definitions already discussed. The impacts statements have been removed, but the impacts are different for 32-bit and 64-bit installations. Subsequent OE definitions have been removed from this discussion of the core configuration file. So have the Languages clauses that follow them.

```
#########################################################
##  Operating Environments
#########################################################

sw_sel "HPUX11i-OE-MC" {
    description = "HP-UX Mission Critical Operating Environment Component"
    sw_source = "core"
    sw_category = "OpEnvironments"
    sd_software_list= "HPUX11i-OE-MC,r=B.11.11.0212,a=HP-
UX_B.11.11_32/64,v=HP"


    (_hp_os_bitness == "32") {
```

```
...
    }
    (_hp_os_bitness == "64") {
...
    }
}

sw_sel "HPUX11i-OE-Ent" {
    description = "HP-UX Enterprise Operating Environment Component"
    sw_source = "core"
    sw_category = "OpEnvironments"
    sd_software_list = "HPUX11i-OE-Ent,r=B.11.11.0212,a=HP-
UX_B.11.11_32/64,v=HP"
    (_hp_os_bitness == "32") {
...
    }
    (_hp_os_bitness == "64") {
...
    }
}

sw_sel "HPUX11i-OE" {
    description = "HP-UX 11i Operating Environment Component"
    sw_source = "core"
    sw_category = "OpEnvironments"
    sd_software_list = "HPUX11i-OE,r=B.11.11.0212,a=HP-UX_B.11.11_32/64,v=HP"
    (_hp_os_bitness == "32") {
...
    }
    (_hp_os_bitness == "64") {
...
    }
}
```

Next is a discussion on the definition of some other software that can be installed.

```
#########################################################
##  Other Software
#########################################################
```

The following definition of Perl from the OE media automatically selects Perl to be installed. The `sw_sel` configuration is initialized to `TRUE`. However, you can deselect the software from installation on the **Software** tab in the Ignite-UX GUI.

```
init sw_sel "perl" {
    description = "Perl Programming Language"
    sw_source = "core"
    sw_category = "OrderedApps"
    sd_software_list = "perl,r=D.5.8.0.B,a=HP-UX_B.11.11_32/64,v=HP"
    impacts = "/opt" 92854Kb
} = TRUE
```

The following software is the Online Diagnostics bundle. This is an example of always-installed software. It is automatically loaded with any `HPUXBaseOS` bundle. You cannot unselect this software bundle from being installed.

```
sw_sel "OnlineDiag" {
    description = "HPUX 11.11 Support Tools Bundle, Jun 2004"
    sw_source = "core"
    sw_category = "HPUXAdditions"
    sd_software_list="OnlineDiag,r=B.11.11.14.15,a=HP-UX_B.11.11_32/64,v=HP"
    impacts = "/usr" 66792Kb
    impacts = "/opt" 943Kb
    impacts = "/var" 12395Kb
    impacts = "/sbin" 80Kb
    impacts = "/etc" 5342Kb
    impacts = "/dev" 1Kb
    load_with_any ~ "HPUXBaseOS" . "*"
}
```

Next is a list of available keyboard layouts. If the system has some hardware controlled by the `ps2` driver, then the keyboards controlled by the `has_ps2` test are added to the list of available keyboards. `Not Applicable` is always a valid choice even if the system does have a keyboard attached.

The follow-on test, `has_usb`, adds more keyboard choices if the system being installed has USB hardware attached to the system. The default is set at the very end to have the keyboard set to "Not Applicable". HP workstations and servers do not have `ps2` and USB keyboards so you can have one or the other but not both types of keyboards connected to a workstation or server.

The available keyboards are shown in the Ignite-UX GUI on the **Basic** tab using the **Keyboards...** button. When you select that button, a list of keyboards available for

selection appears and is generated from this configuration (most servers only show `Not Applicable`).

```
#########################################################
##  Keyboards
#########################################################

_hp_keyboard = { "Not_Applicable" }

has_ps2 {
  _hp_keyboard += {
    "PS2_DIN_Arabic",
    "PS2_DIN_Belgian",
    "PS2_DIN_Belgian_Euro",
    "PS2_DIN_Bulgarian",
    "PS2_DIN_Canada_TBITS-3",
    "PS2_DIN_Canadian_French",
    "PS2_DIN_Czech",
    "PS2_DIN_Czech_Euro",
    "PS2_DIN_Danish",
    "PS2_DIN_Danish_Euro",
    "PS2_DIN_Euro_Spanish",
    "PS2_DIN_Euro_Spanish_Euro",
    "PS2_DIN_French",
    "PS2_DIN_French_Euro",
...
    "PS2_DIN_T_Chinese",
    "PS2_DIN_Turkish",
    "PS2_DIN_UK_English",
    "PS2_DIN_UK_English_Euro",
    "PS2_DIN_US_English",
    "PS2_DIN_US_English_Euro"
  }
}
has_usb {
  _hp_keyboard += {
    "USB_PS2_DIN_Belgian",
    "USB_PS2_DIN_Belgian_Euro",
    "USB_PS2_DIN_Danish",
    "USB_PS2_DIN_Danish_Euro",
```

```
      "USB_PS2_DIN_Euro_Spanish",
      "USB_PS2_DIN_Euro_Spanish_Euro",
…
      "USB_PS2_DIN_Swiss_French2_Euro",
      "USB_PS2_DIN_Swiss_German2",
      "USB_PS2_DIN_Swiss_German2_Euro",
      "USB_PS2_DIN_T_Chinese",
      "USB_PS2_DIN_UK_English",
      "USB_PS2_DIN_UK_English_Euro",
      "USB_PS2_DIN_US_English",
      "USB_PS2_DIN_US_English_Euro"
    }
  }
  init _hp_keyboard = "Not_Applicable"
```

Impacts statements

Previous sections have explained impacts statements in some detail and the circumstances in which they may not accurately represent space usage. This section describes how you might decide to structure your impacts and what level to go with your impacts because such decisions may not be straightforward.

HP-UX installation media is discussed first. The impacts statements on the installation media are all at level one. Consider what limitations this may cause, that is, what problems the following set of (partial) impacts could cause:

```
impacts = "/usr" 687920Kb
impacts = "/opt" 95300Kb
impacts = "/var" 23395Kb
```

Most people encounter no problems. The previous statements assume that you did not create any extra file systems apart from the default file systems that Ignite-UX would create. The impacts do not cause a problem for most people, because Ignite-UX only creates file systems at the top level by default, for example: `/home`, `/var`, `/opt`, and `/usr`.

What happens if you want to create the following extra file systems (for an application)?

```
/opt/application
/opt/application/static
/var/opt/application
/var/opt/application/data
/var/opt/application/tmp
```

Ignite-UX has no information about what data might be stored in these file systems because it has no impacts statements that affect them. Suppose that the impacts are only one level deep and everything is being tracked against `/var` and `/opt` for the application:

```
impacts = "/opt"   1048576Kb
impacts = "/var "  3673088Kb
```

If you create those extra file systems, you might easily misjudge their size. Because the impacts statements have tracked all of the space against `/var` and `/opt`, Ignite-UX cannot know how much space is needed under those file systems so it does not alert you. If your situation is like this, it would make sense to set up your own Ignite-UX server with the correct impacts keywords rather than attempting to use installation media.

Assume that the file systems needed for this application are those in Table 5:

Table 5

Mount point	File System Size (Mb)	Space Used (Mb)
/opt/application/	1024Mb	768Mb
/opt/application/static	512Mb	384Mb
/var/opt/application	2048Mb	1512Mb
/var/opt/application/data	16384Mb	2011Mb
/var/opt/application/tmp	16384Mb	64Mb

Impacts statements for the `sw_sel` clause associated with this application look like the following:

```
impacts = "/opt/application"          786432Kb
impacts = "/opt/application/static"   393216Kb
impacts = "/var/opt/application"      1548288Kb
impacts = "/var/opt/application/data" 2059264Kb
impacts = "/var/opt/application/tmp"  65536Kb
```

The impacts statements place the required impacts at the places where the application (in the environment in which it is installed) has mount points. With this information, Ignite-UX is aware of the impacts associated with each file system.

Creating impacts in this manner (for our optional application with only five impacts statements at up to four levels deep) is a manual job for two reasons:

First, Ignite-UX normally creates impacts at every directory up to the level that you specify. If you have an archive and you use the `archive_impact` command to generate the impacts with the option `-l 4`, the `archive_impact` command lists the space used in every directory up to the fourth level deep. With a large directory structure, this can generate thousands of impacts statements. When Ignite-UX processes a large amount of impacts statements it can slow down to a significant degree.

Combining those large numbers of impacts statements down into a few that match planned mount points is a manual task (and, unfortunately, if the software ever changes you need to do it all again).

Impact Hints

- Keep impact levels to the minimum required to minimize their affect Ignite-UX performance.
- If you need detailed impacts, you can always summarize impacts into the most appropriate form manually (impacts based upon expected mount points) without having to have thousands of impacts statements.

Second, applications may require extra space. In the preceding application, what would you do if the application only installed a few MB into the file system `/var/opt/application/data` but then when configured needed the rest of the space?

You have to plan to have the space there when the application is installed. Sometimes you must manually change impacts statements to increase the impacts on a directory or directory structure because of space requirements on a file system that are not reflected in the size of the software to be installed.

An example of this might be database table spaces that may be required when an application creates a database at initial installation. If this occurs, manually changing impacts statements is the only way to account for the extra space required. Manually increasing the impacts in this case prevents anyone from decreasing the file system size to the point that failures occur.

Categories and other Ignite-UX software attributes

The *instl_adm*(4) manpage states the following about `sw_category`:

```
sw_category cat-tag-string { description = string }
    The sw_category definition provides a grouping mechanism for
    sw_sel's to reference.  The user interface uses the sw_category
    to help the user browse the software more easily.  There are six
    values of cat-tag-string that the user interface treats special.
    Software that are in these groups are represented in special
    locations and by special methods in the user interface.  These
    values are: "HPUXBaseOS", "HPUXEnvironments", "Languages",
    "LanguagesUI", "OpEnvironments", and "UserLicenses" .The only
    attribute associated with a sw_category is a description.
```

The following is what is special about these categories:

- `OpEnvironments`— OEs defined in a configuration are available for selection in the Ignite-UX GUI on the **Basic** tab in the **Environments:** list. They are not selectable using the **Software** tab.
- `HPUXBaseOS`— used for the `HPUXBase32` and `HPUXBase64` bundles that are included in HP-UX B.11.11. In HP-UX B.11.23, there is a single `HPUXBaseOS` bundle that is 64-bit only

(the tag and category have the same name). These bundles also control the setting of `_hp_os_bitness` in a core configuration file.

- `Languages`— in HP-UX 11.00, these pseudo-bundles were extracted from the `HPUXEnvironments` bundles like `HPUXEng64RT`. If you selected "English" and the "64-bits CDE Environment" options, this bundle would be selected. In HPU-UX B.11.11 and later, these were pseudo-bundles that when selected cause `LanguagesUI` category bundles to get installed as a result.
- `LanguagesUI`— these bundles are not shown at all using the user interface; instead, they are automatically selected based upon the bundles with the `sw_category` tag set to Languages. Take, for example, Swedish from the B.11.11 core configuration file:

```
init sw_sel  "Swedish"{
        description="Swedish CDE Environment"
        sw_source="core"
        sw_category="Languages"
} = FALSE


init sw_sel  "CDE-Swedish"{
        description="Swedish CDE Environment"
        sw_source="core"
        sw_category="LanguagesUI"
        sd_software_list="CDE-Swedish,r=B.11.11,a=HP-
UX_B.11.11_32/64,v=HP"
        load_with_all="Swedish"
        impacts="/usr" 70992K
        impacts="/opt" 9098K
        impacts="/sbin" 54K
        impacts="/" 55K
        visible_if=FALSE
        locale={"sv_SE.iso88591:Swedish","sv_SE.utf8:Swedish",
   "SET_NULL_LOCALE:English","ASK_AT_FIRST_BOOT","C:English"}
} = FALSE
```

When Swedish is selected, the bundle `CDE-Swedish` is automatically selected for installation because it has the configuration `load_with_any` set to "Swedish".

- `UserLicenses`— this category is no longer very relevant. HP-UX 11.00 started out with different levels of user licenses. Later, a patch/software bundle was added that provided an unlimited user license to all installations. At HP-UX B.11.11 and later, an unlimited user license is provided as standard.

Defining a custom software configuration

The section of *instl_adm*(4) that defines software configuration is quite long but very useful; only sections of it are discussed here. If you require more information, review the section titled "Software Source and Selections".

Do not attempt to write configuration files for SD depots; instead use the `make_config` command because it correctly defines all configuration data needed. You can then customize the configuration file produced by `make_config`.

The general structure for specifying software is as follows:

```
sw_source src-tag-string
{
source-attributes...
}
sw_category cat-tag-string
{
      description = string
}
sw_sel sel-tag-string
{
      sw_source src-tag-string
      sw_category cat-tag-string
      selection-attributes...
}
init sw_sel sel-tag-string=Boolean
```

Looking at a network recovery `sw_source` and `sw_sel`

To see how software is defined, look at the way that `make_net_recovery` builds a definition of an archive:

```
sw_source "core archive"{
   description = "Recovery Archive"

   load_order = 0
   source_format = archive
   change_media = FALSE

   post_load_script = "/opt/ignite/data/scripts/os_arch_post_l"
   post_config_script = "/opt/ignite/data/scripts/os_arch_post_c"
```

```
    # if nfs_source is used, be sure to export the source.
     (source_type == "NET") {
         nfs_source = "10.0.0.1:/var/opt/ignite/recovery/archives/systema"
     }

    # ftp_source and remsh_source are alternate ways to copy
    # the archive.  An example of the ftp syntax:
    # ftp_source = "anonymous@15.1.54.123:iux"
    # remsh_source = "user@15.1.54.123"
}
```

In the above example, there is a `sw_source` definition called `core archive` that has a description of `Recovery Archive`. The archive is loaded at load order 0 (this is a requirement for a core operating system archive if it is a recovery archive or a golden image; you can only have one archive at load order 0). The attribute `change_media` is set to `FALSE` because you do not want to be prompted to change the media. It does not make sense to change media when the source is over the network using NFS.

You then test to see if the `source_type` is "`NET`" and if it defines the `nfs_source` to point to the archive.

In the comments regarding other network access methods, .the `source_type` variable defaults to the type of media that the system was booted from (although this automatically becomes "`NET`" when an Ignite-UX server is being used, regardless of whether the system was booted from disk or tape).

Next, categories are defined because you are going to have to provide one `sw_sel` in the `Languages` category and another in the `HPUXEnvironments` category:

```
##########################################################
##  Software Categories
##########################################################

sw_category "Languages" {
   description = "Languages"
}

sw_category "HPUXEnvironments"{
    description = "HP-UX Operating Environments"
}
```

You can now define a `sw_sel`, called "`golden image1`", to hold the definition of the `Recovery Archive`. The `init` command is used to initialize the `sw_sel` and at the end

of the definition it is set to `TRUE`. This allows the Ignite-UX GUI to change this value (although changing the value in this case does not make sense).

You then define the `sw_source`; the `sw_source` appears earlier in this configuration file. The `sw_category` for the `sw_sel` is set to `HPUXEnvironments`. This so that the description `Recovery Archive` appears in the **Environments:** selection list in the Ignite-UX GUI so it can be easily selected. The `archive_type` is then defined, in this case it is a gzipped `tar` file (refer to *`instl_adm`*`(4)` for other valid types).

Next, the archive path relative to the NFS mount point in the `sw_source` is given. The recovery archive is only visible if the system can run 64-bit HP-UX. If the `sw_sel "golden image1"` is selected, set `_hp_os_bitness` to 64, and then set the variable so it cannot be changed using the **Additional** button on the **Basic** tab of the Ignite-UX GUI.

```
#########################################################
##  Operating Environments
##
#########################################################

#############################################################
##  64-bit OS archives
#############################################################
init sw_sel "golden image1" {
   description = "Recovery Archive"

   sw_source = "core archive"
   sw_category = "HPUXEnvironments"
   archive_type = gzip tar
   # For NFS, the path to the archive is relative to the mount point
   # specified in the sw_source:

    (source_type == "NET") {
        nfs_source = "10.0.0.1:/var/opt/ignite/images"
    }

...
   visible_if  = can_run_64bit
} = TRUE

(sw_sel "golden image1") {
   _hp_os_bitness = "64"
}
_hp_os_bitness visible_if false
```

Next, a language is set up so there is a choice of no locale English, or C locale and English. Your choice depends on the languages that would be available depend on the system at the time the recovery archive was created:

```
sw_source "no select" {
   source_format = cmd
}

init sw_sel "English" {
   description = "English Language Environment"
   sw_source = "no select"
   sw_category = "Languages"
   locale = { "SET_NULL_LOCALE:English", "C:English" }
} = TRUE
```

Using a `sw_sel` to run commands instead of installing software

The following configuration acts in a similar way to the language selection earlier. The `source_format` of `cmd` in the `sw_source` definitions does not cause any software to be loaded. The load order statement forces the command to run after everything else (in case someone does anything at the default load order of 5). The `source_format` of `cmd` gives you a great deal of flexibility in running scripts.

You have to define any `sw_category` that you want to use. Before you can use a software category, you must define a `sw_category`. This is not true for the `sw_category` "Uncategorized"; this `sw_category` is applied to any software that does not explicitly define a `sw_category`. If all software is given, the **Uncategorized** `sw_category` will not appear on the **Software** tab in the Ignite-UX GUI. You should not explicitly assign a `sw_category` **Uncategorized** to any software. In the following example, the `sw_category` of `"SiteSpecific"` is defined.

The `sw_sel`, however, defines a `post_load_cmd` that runs:

```
sw_category "SiteSpecific" {
        description = "Local Site Commands"
}

sw_source "site commands" {
   source_format = cmd
   load_order = 10
}

init sw_sel "Run site commands" {
```

```
    description = "Run site commands after OS loaded"
    sw_source = "site commands"
    sw_category = "SiteSpecific"
    post_load_script="/var/opt/ignite/scripts/run_after_load"
} = TRUE
```

The difference between this method and placing a `post_load_cmd` into the core operating system load `sw_sel` is that you have the opportunity to prevent the commands from running the Ignite-UX GUI by deselecting the software (even though in this case there really is no "software").

However, be aware that any script you run must be either accessible using `tftp`, if this is a network installation, or located in the `SCRIPTS` file, if the installation is happening from media (in the LIF produced by `make_medialif`).

If you want to run a script that has already been installed with other software, use a `cmd` hook instead of a script hook:

```
init sw_sel "Run site commands" {
    description = "Run site commands after OS loaded"
    sw_source = "site commands"
    sw_category = "SiteSpecific"
    post_load_cmd="/opt/site/bin/run_after_load"
} = TRUE
```

You must ensure that the load order associated with the `sw_source` prevents the "`cmd`" `sw_source` from running until after any software that it depends on has been loaded (if it has dependencies).

Using a `sw_sel` to apply kernel parameters

You can use a `sw_sel` clause to control the application of kernel parameters to a system, as well. The Ignite-UX configuration items that can be used to change kernel parameters (from the *`instl_adm`*`(4)` manpage) are as follows:

```
mod_kernel = cplx-string
mod_kernel += cplx-string
    This keyword can be used to add drivers or tunable parameters to
    the system's kernel that is built during the Ignite-UX process.
    The format of cplx-string may be either "driver" or "tunable
    value".

    The largest of any tunable parameter that exists in either the
    /stand/system file or that is specified will be used.  Beginning
    with the 11.23 release, if a tunable is not found in
```

/stand/system, then it will also be compared with the default value as reported by kctune. Ignite-UX does not compare the values of formulas to discrete numerical values, or two formulas, or hexadecimal values, in order to determine which is larger. It will issue a note message stating that it will assume the last mod_kernel keyword parsed is larger (regardless of whether it is a formula or discrete numerical value) and will apply it. There is no bounds checking done on tunable parameters.

Decimal values should be in the range from 0 to 2147483647 (2"31-1). The shell that is used to evaluate these values uses signed 32-bit arithmetic for decimal values, and it has problems when values exceed this limit. If larger values are needed, convert them to hexadecimal since the shell does not evaluate them.

Note that any allowable syntax supported by the config command can be used in mod_kernel keywords. This includes formulas. For example, it is possible to use:

```
mod_kernel += "ninode (20+8*MAXUSERS+NPTY+" + ${"%d" _inc} + ")"
```

assuming _inc has been initialized to some value in the configuration file.

The = operator will override any prior global mod_kernel assignments. The += operator will add to any prior settings. Notice that mod_kernel statements may also be associated with a sw_sel definition. The = operator does not have any effect on mod_kernel assignments made in a sw_sel.

Beginning with the 11.23 release, the format for cplx-string arguments was enhanced. There are two new formats specifically that are understood (in addition to the older formats). They are:

```
mod_kernel += "tunable name value"
mod_kernel += "module name [state]"
```

The former is the same as how tunables were handled before only the keyword tunable is put at the beginning. The latter is how

```
    kernel modules are added to the system.  The optional state
    argument has one of four values: static, auto, loaded and best.
    No checking is performed on this value. See kcmodule(1M) for more
    information.

    Actions involving mod_kernel are done before those for both
    set_kernel and rm_kernel.

set_kernel = cplx-string
set_kernel += cplx-string
    This keyword is the same for drivers as the mod_kernel keyword in
    that it will add the driver to the kernel.  For tunables it
    differs from mod_kernel in that it will set the tunable to the
    arbitrary value as defined by cplx-string.  No comparisons or
    checks are performed with prior or default values.  Actions
    involving set_kernel are done after those for mod_kernel but
    before those for rm_kernel.

rm_kernel = cplx-string
rm_kernel += cplx-string
    This keyword will remove the driver or tunable from the
    /stand/system file.  For drivers, this implies the driver will be
    removed from the kernel.  For tunables, this implies that the
    tunable will revert back to its default value.  Actions involving
    rm_kernel are done after those for both mod_kernel and
    set_kernel.
```

The following example shows that a new category is created, called `KernelConfig`, which defines sets of kernel parameters that can be applied to a system. Ignite-UX does not do any parameter validation. That is followed by the "no-op" `sw_source` clause:

```
sw_category "KernelConfig" {
        description = "Kernel Configuration sets"
}

sw_source "Kernel Configuration" {
   source_format = cmd
   load_order = 11
}
```

The first `sw_sel` defines a set of kernel parameters that can be applied to a system when Java is installed. The only new items here are the `exrequisite` statement and

set_kernel. The set_kernel statements enable you to easily set the formula as the value to be assigned to a kernel tunable since Ignite-UX does not attempt to check the value being assigned to the kernel tunable. It is simply set to that value.

The exrequisite ensures that if this sw_sel is selected, none of the exrequisites are also selected (they are unselected when this sw_sel is selected).

```
sw_sel "Kernel Config for Java" {
    description = "Tunable settings required for Java 1.4.2"
    sw_source = "site commands"
    sw_category = "KernelConfig"
    exrequisite = "Kernel Config for Servers"
    exrequisite += "Kernel Config for Workstations"
    set_kernel = "maxusers 400"
    set_kernel += "max_thread_proc maxusers*3"
    set_kernel += "maxfiles 2048"
    set_kernel += "maxfiles_lim 2048"
    set_kernel += "ncallout 2*((((nproc*7)/4)+16)*2)"
    set_kernel += "nkthread 2*max_thread_proc"
    set_kernel += "nfile (2*nproc)+1000"
    set_kernel += "nproc ((maxusers*5)+64)"
}
```

The next two sw_sels reinforce the use of exrequisites. Because all the sw_sel clauses explicitly state that if one is selected the other two cannot be, only one of them can be selected at any time.

```
sw_sel "Kernel Config for Servers" {
   description = "Tunable settings required for Servers"
   sw_source = "site commands"
   sw_category = "KernelConfig"
   exrequisite = "Kernel Config for Java"
   exrequisite += "Kernel Config for Workstations"
   set_kernel = "maxdsiz 0x4000000"
   set_kernel += "maxusers 128"
}
sw_sel "Kernel Config for Workstations" {
   description = "Tunable settings required for Workstations"
   sw_source = "site commands"
   sw_category = "KernelConfig"
   exrequisite = "Kernel Config for Java"
   exrequisite += "Kernel Config for Servers"
   set_kernel = "maxdsiz 0x4000000"
   set_kernel += "maxusers 96"
}
```

In the following example, you have a configuration that to apply to sw_sel so it will automatically select which sw_sel (containing the kernel configuration changes) to apply. If the sw_sel "B9789AA, r=1.3.1.09.08" is selected, then the sw_sel "Kernel Config for Java" will be automatically selected; otherwise, if this is a server

selected then, "Kernel Config for Servers" will be selected. If neither of these are selected, the "Kernel Config for Workstations" will be selected.

```
( sw_sel "B9789AA,r=1.3.1.09.08" ) {
   init sw_sel "Kernel Config for Java" = TRUE
} else {
    ( HARDWARE_MODEL ~ "9000/8" | MODEL ~ "ia64.*server" ) {
         init sw_sel "Kernel Config for Servers" = TRUE
    } else {
         init sw_sel "Kernel Config for Workstations" = TRUE
    }
}
```

It is important to understand that, starting at HP-UX B.11.23, the format of the cplx-string arguments to mod_kernel, set_kernel, and rm_kernel have changed and are incompatible with the previous format. The following information is from the *Ignite-UX Reference*:

"Beginning with the B.11.23 release, the format for cplx-string arguments is enhanced. There are additional formats:

```
mod_kernel += "tunable name value"
mod_kernel += "module name [state]"
```

The first performs the same as it did previously with the addition of the keyword tunable. The second describes how kernel modules are added to the client. The optional state argument has one of four values: static, auto, loaded and best. No syntax checking is performed for this value. See *kcmodule*(1M) for more information."

So to tune maxdsiz at HP-UX B.11.11, you would need to set it as follows:

```
set_kernel = "maxdsiz 0x4000000"
```

At HP-UX B.11.23, you would need to use:

```
set_kernel = "tunable maxdsiz 0x4000000"
```

Under a lot of circumstances configuration files are not shared between HP-UX revisions; so this does not create an immediate problem. However, if you do share a configuration file that tunes the kernel between HP-UX releases, you should probably do it as follows. This example assumes that you wish different tunables for HP-UX B.11.00, B.11.11, and B.11.23:

```
...
( release == "B.11.00" ) {
    set_kernel = "maxdsiz 0x4000000"
    set_kernel += "maxusers 64"
} else {
```

```
    ( release == "B.11.11" ) {
        set_kernel = "maxdsiz 0x4000000"
        set_kernel += "maxusers 96"
    } else {
        ( release == "B.11.23" ) {
            set_kernel = "tunable maxdsiz 0x4000000"
        } else {
            error +="release " + ${release} + " not supported by this
            configuration."
        }
    }
}
...
```

A configuration file shared between only HP-UX B.11.00 and B.11.11 does not need to be written this way. It is only needed when you adjust kernel tunables in the one configuration for HP-UX B.11.23 (and future releases of HP-UX) and HP-UX B.11.11 and B.11.00.

Forcing software (`sw_sel`) clauses to be installed

Be careful when marking software automatically for installation using a configuration file. If you mark it in the following way (set it directly equal to `TRUE`), the software is marked as required in the Ignite-UX GUI and you cannot manually unselect the software:

```
sw_sel "Kernel Config" {
   description = "Tunable settings"
   sw_source = "site commands"
   sw_category = "KernelConfig"
   set_kernel = "maxdsiz 0x4000000"
   set_kernel += "maxusers 96"
}=TRUE
```

Compare the previous software clause to the following. Because it starts with `init` you can change its state using the Ignite-UX GUI during installation, although by default it is marked for installation.

```
init sw_sel "Kernel Config" {
   description = "Tunable settings"
   sw_source = "site commands"
   sw_category = "KernelConfig"
   set_kernel = "maxdsiz 0x4000000"
   set_kernel += "maxusers 96"
}=TRUE
```

You can force software installation in other ways such as with the keywords `load_with_any` and `load_with_all`. However, you should not combine usage of the `init` keyword with these methods.

Keep this in mind when defining software because you do not want to force software onto a system unintentionally.

Automating dependencies in software

This section explains the ways of enforcing dependencies between software with Ignite-UX.

`Corequisites` are used to enforce dependencies between software that you want to load at the same time:

```
corequisite [+]= tag-string

    Indicates that the sw_sel referred to by tag-string should
    be loaded along with this sw_sel.  Multiple corequisites may
    be listed using one corequisite statement per, and by using
    the += operator.
```

In the following example, there are the `sw_sel "Product XYZ"` and `"Product XYZ - 4GL"`. The 4GL product requires the other product to be installed as well, so introduced here is a corequisite for the 4GL to install automatically `"Product XYZ"` when it is selected for installation.

```
sw_sel "Product XYZ" {
    description = "Product XYZ base runtime"
    sw_source = "Applications for 11i Version 1"
    sw_category = "SiteApps"
    sd_software_list = "ProductXYZ,r=4.0,v=XYZCorp"
}

sw_sel "Product XYZ - 4GL" {
    description = "4GL option for Product XYZ"
    sw_source = "Applications for 11i Version 1"
    sw_category = "SiteApps"
    corequisite = "Product XYZ"
    sd_software_list = "ProductXYZ4GL,r=4.0,v=XYZCorp"
}
```

See "Using a `sw_sel` to apply kernel parameters" for `exrequisite` examples.

```
exrequisite [+]= tag-string
```

Defines an exclusive relationship between the current sw_sel and the one referenced by tag-string. This will prevent the referenced sw_sel from being selected any time that this sw_sel is selected (and vice-versa). The += operator may be used to define multiple exrequisites.

Instead of automatically selecting something for installation based upon the name of a sw_sel clause, exrequisite instead works on the name of a sw_category[46]. This form could be used to set up two sets of incompatible software in different software categories and to use the name of the sw_category as an exrequisite to software in the other category.

exrequisite = sw_category

Specifies that this sw_sel is to be exclusive with all other software in the same category. This attribute can be used in addition to other exrequisites and does not override them even though the += operator is not used in this case.

The load_with_any configuration item gives you flexibility in automatically selecting software to be installed, based upon what other software may be selected for installation. By using the = sign with load_with_any, you specify that when any of the listed sw_sel clauses are selected, this sw_sel is selected also.

load_with_any = tag-string [[| tag-string]...]

Specifies that when any of the one or more sw_sels listed are selected, that this sw_sel should be selected for loading as well. Multiple tags may be listed separated by the | character.

The following form enables regular expression matching (not EREs, only those allowed by *fnmatch*(3)) to select the software or category that software should be loaded with.

load_with_any ~ tag-regexp [[| tag-regexp]...]

load_with_any ~ category.tag-regexp ...

Similar to above, except that when the ~ operator is used instead of =, then the sw_sel tags listed are treated as

[46] Be careful when creating sw_category names that are the same as sw_sel names. Using exrequisites can cause Ignite-UX to produce unexpected results if you use the same name for an existing sw_sel and sw_category.

fnmatch(3C) regular expressions that can be used to match any selection fitting the given pattern. (Note that this operator uses fnmatch(3C) pattern matching expressions. Other uses of the ~ operator in the configuration file, uses extended regular expressions. This difference is for compatibility with some existing data.) The category string can be specified (with a "." separator) in order to limit the matches to those selections in the specified software category.

The following example uses expression matching:

```
sw_sel "perl" {
    description = "Perl Programming Language"
    sw_source = "core"
    sw_category = "OrderedApps"
    sd_software_list = "perl,r=B.5.6.1.C,a=HP-UX_B.11.11_32/64,v=HP"
    impacts = "/opt" 49480Kb
    load_with_any ~ "HPUXBaseOS" . "*"
}
```

The preceding `load_with_any` automatically selects `perl` to be loaded when any `sw_sel` with a category of `HPUXBaseOS` is selected.

The `load_with_all` configuration item enables you to specify a list of `sw_sel` clauses that must all be selected automatically to select a `sw_sel` for installation:

```
load_with_all = tag-string [[ & tag-string ]...]
    Specifies that when all the sw_sels listed are selected,
    then this sw_sel should be selected as well.  Multiple tags
    may be listed separated by the & character.

    If multiple load_with_* keywords are specified, the list of
tag-strings will be added to any already listed for the
respective keyword.
```

The `load_with_*` configuration items automatically unselect `sw_sel` clauses as appropriate when you are changing what `sw_sel` clauses are selected for installation (that is, enforce dependencies). There is one exception to this rule: if you manually select a `sw_sel` for installation and it has any `load_with_*` configuration items that are not satisfied, it cannot be automatically unselected because it was manually selected.

Installing patches

Patches are best installed using software configuration rather than command hooks like "post_load_cmd" because a new kernel is built after all software is installed. If you install patches using command hooks, a new kernel is not built, which may cause some unusual behavior in HP-UX where commands depend on kernel functionality that is not in the kernel.

Configuration for volume and disk groups

In this section, the default configuration file for an HP-UX release is assumed always to be included before any others. Because internal variables defined by that configuration file must be set for Ignite-UX to work correctly.

Overview

The general format for Disk Group, Volume Group, and whole disk layout is as follows:

```
partitioned_disk
{
    physical_volume disk[hw_path|index]
    {
        physical_volume-attributes...
    }
    fs_partition
    {
        file-system-attributes...
    }
    swap_partition
    {
        swap-attributes...
    }
}
volume_group group_name
{
    volume_group-attributes...
    physical_volume group "pvgname"
    {
        physical_volume disk[hw_path|index]
        {
        physical_volume-attributes...
    }
```

```
            ...
    }
    ...
    physical_volume disk[hw_path|index]
    {
        physical_volume-attributes...
    }
    ...
    logical_volume
    {
        file-system/swap-attributes...
    }
    logical_volume "lvname"
    {
        file-system/swap-attributes...
    }
    ...
}
```

Note that partitioned disks are not used very often in more modern systems as the size of the boot disk cannot be more than approximately 2GB in size on PA-RISC systems. There is no such restriction on Itanium®-based systems.

Before looking at anything in detail, the attributes that can be set in each of the preceding examples are introduced in the following configuration examples. For more details on the individual attributes, refer to *instl_adm*(4).

Configuration examples

The following section presents several different disk configuration examples. They may prove useful when developing your own configurations.

This section does not discuss the extra space that may be allocated to file systems because of impacts configuration statements or the effect that variables such as `_hp_addnl_pct_free` or `_hp_efi_partition_size` may have.

Example one (custom disk layout)

This example defines a configuration that is dependent on whether or not a particular software product is installed:

```
#
# This config file builds on top of the default config file for
# an OS release since it reuses some of the same variables.
```

```
# It should be included into a cfg clause in the index file after
# the default config file for a release.  The following software
# should be defined in any clauses that includes this file:
#   "Product XYZ"
#      "Product XYZ - 4GL"
#
```

Remember that if you change the disk configuration, the disk layout name is changed to "Modified LVM Layout". Consequently, things defined in the configuration may no longer be evaluated anymore. For example, you can set the name of the volume group in the configuration in this example using the **Additional** button on the **Basic** tab in the Ignite-UX GUI (see `_my_volume_name` later in this section). However, once you modify some part of the disk layout, the name of the layout becomes "Modified LVM Layout" and changing the value of `_my_volume_name` no longer has any effect.

The name is defined by using the selection list produced by the **Additional** button on the **Basic** tab in the Ignite-UX GUI. For the selection, "Name of the root volume group" you can select `vg00` or `vgroot`. However, you can also enter a new name into the field and that will be used (since it was not defined as an `enum` you cannot enforce a choice from the available names).

The name can be changed on the **Additional** button. However, it only takes effect when the disk layout selected is "Custom configuration for Product XYZ".

```
#
# Allow the user to change the name of the root volume group
#
_my_volume_name = { "vg00" , "vgroot" }
init _my_volume_name = "vg00"
_my_volume_name help_text "Name of root volume group"
_my_volume_name visible_if TRUE
```

Here a new disk layout name is being added to the configuration, not setting it as one of the choices. This configuration file builds on top of the existing default operating system release configuration file. It is not a replacement because many variables need to be defined that are not worth the effort to set.

```
#
# Add a new disk layout type
#
_hp_disk_layout += { "Custom configuration for Product XYZ" }
```

The test on `sw_sel Product XYZ` to the variable set `_hp_disk_layout` to `Custom configuration for Product XYZ` is only evaluated once. If `Product XYZ` is not

selected for installation by the time this configuration file is parsed, you must manually change the disk layout to `Custom configuration for Product XYZ` during installation. That is, after selecting the software products that the disk configuration depends on, if they are to be installed, to make sure that the optional logical volume is created.

To make the disk configuration modifiable, remember to use `init` before the `_hp_disk_layout`.[47]

```
#
# Automatically select the disk layout if product XYZ is being
# installed.
#
(sw_sel "Product XYZ") {
        init _hp_disk_layout = "Custom configuration for Product XYZ"[48]
}
```

Next is the definition of the disk layout `Custom configuration for Product XYZ`:

```
( _hp_disk_layout == "Custom configuration for Product XYZ" ) {
```

First, make sure that all disks are greater than 36GB. An error will be issued if there are smaller disks because this disk layout is only intended for disks 36GB or higher.

Note:
This prevents the user from proceeding with an installation (or force a non-interactive installation to become interactive) if the size of the root disk is less than what is being tested for[49].

Looking for errors where possible can help ensure that a configuration does not allow someone to do something that cannot work.

```
(disk[_hp_root_disk].size < 34000Mb) {
        error+="This configuration cannot be used with disks of less than
36Gb"
```

[47] Without "`init`" in front of the `_hp_disk_layout` variable, the Ignite-UX GUI cannot make changes to the disk configuration since all changes (in this case) are saved into the disk layout called "Modified LVM Layout". If the Ignite-UX GUI cannot change `_hp_disk_layout`, then the name of the disk layout cannot be changed and no file system changes asked for on the file system tab are honored.
[48] You cannot not have an effects relationship between a variable and a software selection (or a use model). If this software product is selected and the disk layout is set to "Custom configuration for Product XYZ", unselecting the software does *not* cause the disk layout to be reevaluated.
[49] Keep in mind that GB for disks is not 36*1024*1024*1024 (38654705664) bytes but approximately 36*1000*1000*1000 (36000000000) bytes, a difference of over 2GB. This test, to verify whether the size is less than 34000MB (35651584000 bytes), is approximately correct.

```
    }
```

You can now you start the definition of the volume group that has its name set by the variable `_my_volume_group`; the volume group is explicitly defined to be the LVM:

```
volume_group _my_volume_name
{
          #
          # Only allow LVM
          #
          usage = LVM
```

You now define some volume group attributes, specifically the maximum number of physical extents (Ignite-UX changes this to suit the disks as required). In addition, you set up the PE size here as well. The default configuration file sets the variable up `_hp_root_disk` for you.

```
#
# Setup for defaults allowing of 36 and 72Gb disks as boot disk
#
max_physical_extents = 2500
(disk[_hp_root_disk].size < 40000MB) {
          physical_extent_size=8
} else {
          physical_extent_size=16
}
#
# The root disk was found by Ignite-UX for us
#
physical_volume disk[_hp_root_disk]
```

Now you define the `root` file system. In all of these file systems, you are not giving them explicit names; this is left up to Ignite-UX. Here, you only set the minimum number of fields for the `root` file system attributes:

```
#
# Define the root file system
#
logical_volume
{
```

```
        mount_point = "/"
        usage=VxFS
        size=400Mb
        contiguous_allocation = true
        bad_block_relocate    = false
}
```

Next is primary swap and dump. This configuration does not define a secondary swap. The size is interesting; allocate at least `_hp_min_swap` as the size. The variable `_hp_min_swap` again is set by the default configuration file. The maximum size is `_hp_pri_swap`; however, depending on the available disk space, the actual amount of space allocated is somewhere between `_hp_min_swap` and `_hp_pri_swap`.

```
#
# Define the swap/dump
#
logical_volume
{
        mount_point = "primary"
        usage=SWAP_DUMP
        contiguous_allocation = true
        bad_block_relocate    = false
        size = _hp_min_swap | remaining | _hp_pri_swap
}
```

Currently, for PA-RISC systems, the boot file system *must* be HFS. For Itanium®-based systems, the boot file system can be either HFS or VxFS. So here, you define a fixed-size boot file system (`/stand`) to be HFS. HFS is the most compatible, although you could add tests to see if the system is Itanium®-based or PA-RISC and then change the file system type based upon that information.

```
#
# Define /stand
#
logical_volume
{
        mount_point = "/stand"
        usage=HFS
        blksize = _hp_HFS_blksize
        fragsize = _hp_HFS_fragsize
        contiguous_allocation = true
```

```
        bad_block_relocate    = false
        size = 300Mb
}
```

Next, you define `/tmp`; the size is predicated on the current selection state of `Product XYZ – 4GL` at the time the configuration is parsed. If the software is selected when the configuration is parsed, the size is different from when it is not selected. If the software changes state after this disk layout is chosen, the size does not change because of it.[50]

Important:
Size is only defined based upon software selection when the configuration is initially parsed.

If the product `Product XYZ – 4GL` is selected, the size is a minimum 100MB and up to 400MB depending on the remaining disk space. If the product `Product XYZ – 4GL` is not selected, the size is a minimum 50MB and up to 200MB depending on the remaining disk space.

```
#
# Define /tmp
#
logical_volume
{
        mount_point = "/tmp"
        usage=VxFS
        (sw_sel "Product XYZ - 4GL" ) {
              size = 100Mb | remaining | 400Mb
        } else {
              size = 50Mb | remaining | 200Mb
        }
}
```

The next file system is `/usr`. The minimum size is 500MB. Depending on available disk space, the volume may be expanded to require a minimum of 20 percent free space after the file system impacts statements associated with software being installed into `/usr` are taken into account.

Minimum free percentages of disk space may be a good compromise for file systems in systems where you know how much space is needed and you only require some additional overhead space to be left available (this is true for file systems such as `/usr` and `/opt`).

[50] You cannot have effects relationships between variables and software selections and use models.

```
#
# Define /usr
#
logical_volume
{
          mount_point = "/usr"
          usage=VxFS
          size = 500Mb | remaining | 20% free
}
```

The `/var` file system has a different definition for its size. It is whatever space can be allocated up to 1000MB plus the size of `_hp_pri_swap`. Therefore, the final size for this file system depends on the free disk space once minimum sizes have been allocated to file systems.

```
#
# Define /var
#
logical_volume
{
          mount_point = "/var"
          usage=VxFS
          size = remaining | 1000Mb + _hp_pri_swap
}
```

The `/var/opt/pxyz` file system is only defined to be created if the software product `Product XYZ – 4GL` is selected for installation when this disk layout is selected. This is similar to the previous size tests for `/tmp`. This file system is not defined if the product `Product XYZ – 4GL` is selected for installation after the disk layout has been changed to `Custom configuration for Product XYZ`. Similarly, the file system is not removed from the disk layout if the product `Product XYZ – 4GL` is unselected unless the disk layout is changed and then changed back to `Custom configuration for Product XYZ`. Of course, it could be manually removed.

```
#
# Define /var/opt/pxyz
#
(sw_sel "Product XYZ - 4GL" ) {
    logical_volume
    {
```

```
        mount_point = "/var/opt/pxyz"
        usage=VxFS
        size = remaining | 2000Mb
    }
}
```

The last volumes are defined as follows:

```
#
# Define /var/tmp
#
logical_volume
{
        mount_point = "/var/tmp"
        usage=VxFS
        size = remaining | 1000Mb
}
#
# Define /opt
#
logical_volume
{
        mount_point = "/opt"
        usage=VxFS
        (sw_sel "Product XYZ - 4GL" ) {
             size = remaining | 1500Mb
        } else {
             size = remaining | 20% free
        }
}
#
# Define /home
#
logical_volume
{
        mount_point = "/home"
        usage=VxFS
        size = 500Mb
}
```

The default configuration files make setting up a disk configuration much more complex than for a relatively simple disk configuration.

To change the layout of this root volume group to be VxVM, you must set the name of the disk group to `rootdg`; instead of giving the user a choice of names, change `usage = LVM` to `usage = VXVM`.

Example two (selection of disk layout based on hardware)

The configuration example in this section shows a fragment of a disk layout definition that selects the disk layout based upon the model string and the size of the disk you are installing to (you do not need to worry about the disk layout itself).

You must first add the new disk layout names into `_hp_disk_layout`. Remember to add them as choices because you want the user to be able to select them using Ignite-UX GUI:

```
#
# Add new disk layout types
#
_hp_disk_layout+= { "Custom configuration rp8400 w/18Gb disks",
                    "Custom configuration rp8400 w/36Gb disks",
                    "Custom configuration rp8400 w/72Gb disks",
                    "Custom configuration rp8400 w/>100Gb disks" }
```

The following test makes sure that the root disk size is at least the minimum required size for installing an rp8400 system[51]. If the root disk is too small, an error message is presented to users to prevent them from being able eventually to select **Go!** in the Ignite-UX GUI and install the system. The only recourse is to change the disk layout or increase the size of the root disk by selecting a new one.

```
(hardware_model == "9000/800/SD16K-A" & disk[_hp_root_disk].size < 16000Mb ) {
        error += "This configuration does not support installing disks of
<18Gb please choose another disk to install to that is at least 18Gb in size"
}
```

The following test determines whether the disk layout should be changed to `Custom configuration rp8400 w/18GB disks`. A similar test is done for all of the other disk layouts that you can set.

```
(hardware_model == "9000/800/SD16K-A" & disk[_hp_root_disk].size >= 16000Mb &
        disk[_hp_root_disk].size < 30000Mb ) {
        init _hp_disk_layout = "Custom configuration rp8400 w/18Gb disks"
```

[51] The rp8400 is a marketing name for a system. The model string of the system is actually 9000/800/SD16K-A.

```
}

(hardware_model == "9000/800/SD16K-A" & disk[_hp_root_disk].size >= 30000Mb &
         disk[_hp_root_disk].size < 60000Mb ) {
         init _hp_disk_layout = "Custom configuration rp8400 w/36Gb disks"
}

(hardware_model == "9000/800/SD16K-A" & disk[_hp_root_disk].size >= 60000Mb &
         disk[_hp_root_disk].size < 100000Mb ) {
         init _hp_disk_layout = "Custom configuration rp8400 w/72Gb disks"
}

(hardware_model == "9000/800/SD16K-A" & disk[_hp_root_disk].size >= 100000Mb )
{
         init _hp_disk_layout = "Custom configuration rp8400 w/>100Gb disks"
}
```

Next, you start the `Custom configuration rp8400 w/18GB disks` disk layout definition. The tests that you perform inside the definition prevent anyone from misusing the defined layout for the wrong system type and disk size. These tests reinforce the previous default selection code.

```
(_hp_disk_layout == "Custom configuration rp8400 w/18Gb disks") {
    volume_group "vg00"
    {
         usage = LVM

         (hardware_model != "9000/800/SD16K-A") {
              error += "This disk layout is for use with a rp8400"
         }
         (disk[_hp_root_disk].size < 16000Mb |
              disk[_hp_root_disk].size > 30000Mb ) {
              error += "Disk size not between 16000Mb and 30000Mb"
         }
...
```

You leave the custom configuration at that point since you are only looking for some other tests for automating selection and enforcing it with tests and the `error` keyword. Instead of using `error`, using `warning` might alert the user to choose the selection when the tests fail (for example, to use it on an rp7410 instead).

Example three

The example in this section is intended for use in an installation file system not in configuration files. The best use of this configuration is in an installation file system contained in LIF and used to boot a system using tape | CD | DVD so it does not use DHCP to gain an IP address. This section does not discuss other configuration items you might set like `_hp_lanadmin_args`.

This example is intended for use with PA-RISC systems. For information on how to achieve these same results with Itanium®-based systems (or to create media that can be used on both hardware architectures), see "Building an Installation DVD" in the *Ignite-UX Administration Guide*, Edition 18 or later. Additionally, this section contains information about the placement of a LIF inside an El-Torito boot image for use with Itanium®-based systems.

Part A (custom configuration in installation file system)

This part of the example deals with simply providing an IP address and network information that can be applied to an LIF (and hence to the installation file system within the LIF). This method requires you to build a different LIF for every system and place it onto media separately.

The following configuration is needed to contact the Ignite-UX server:

```
server="10.0.0.2"
netmask[]="255.255.255.0"
route_gateway[0]="10.0.1.1"
route_destination[0]="default"
disable_dhcp=true
```

This line gives the system its IP address:

```
ip_addr[]="10.0.1.45"
```

Doing this may not be very useful on multi-homed hosts because the LAN interface at the lowest hardware address is given the IP address. To give the IP address to a specific LAN interface you must do the following:

```
ip_addr[0/0/0/0]="10.0.1.45"
```

This is applied to an LIF (created with `make_medialif -m`) with the `instl_adm` command. The LIF can then be placed on media.

Note:
Unless you are using unallocated IP addresses, a boot tape or CD | DVD created in this way is specific to a system. You do not

want to boot another system with it while the IP address is in use on a live system.

Part B (Installation file system custom network config)

This part of the example deals with providing a more generic (for example, can be used on multiple systems and has information unique to each) solution for IP address and network information that can be applied to an LIF (and therefore to the installation file system within the LIF).

The following section would be generic to all systems located on one LAN segment; they can all access the same gateway:

```
server="10.0.0.2"
netmask[]="255.255.255.0"
route_gateway[0]="10.0.1.1"
route_destination[0]="default"
disable_dhcp=true
```

For the system-specific parts, each of the definitions of the IP address is protected by a test on the MAC address:

```
( lla[0/0/0/0] == "080009654321" ) {
        ip_addr[0/0/0/0]="10.0.1.45"
}
( lla[2/0/2] == "080009654123" ) {
        ip_addr[2/0/2]="10.0.1.46"
}
( lla[1/0/0/0] == "080009654213" ) {
        ip_addr[1/0/0/0]="10.0.1.47"
}
...
```

This enables the tape or CD|DVD to be used on all systems within one subnet.

Configuration parameters in the installation file system

This section discusses configuration parameters that are most suited to being placed in an installation file system using `instl_adm`. The most likely reason that you would place them in the installation file system would be that they need to be seen by Ignite-UX to control the way that Ignite-UX works before it attempts to mount the client's directory from an Ignite-UX server (in a network installation).

For more information regarding the configuration parameters discussed in this section, see "Configuration examples" and "Example three" in this document. In addition, refer to *instl_adm*(4).

Networking

The information in this section only applies to the usage of Ignite-UX after the installation kernel has started running AND after the installation file system has been loaded (since the configuration is stored in the first 8KB of the installation file system).

The following networking configuration parameters are covered elsewhere in this document. For more information, see the following:

```
_hp_lanadmin_args (see "_hp_lanadmin_args")
_hp_nfs_mount_opts (see "_hp_nfs_mount_opts")
_hp_nfs_mount_retries (see "_hp_nfs_mount_retries")
_hp_tftp_cmds (see "_hp_tftp_cmds")
```

The following networking options are best set within an installation file system since Ignite-UX must be aware of them either before it contacts the Ignite-UX server or when it is attempting to start networking.

The DHCP configuration parameters are mentioned here because Ignite-UX requests an IP address using DHCP when it starts and before it first attempts to make contact with the Ignite-UX server.

- `dhcp_class_id`—this configuration item can be used "as is" (for example, applies to every LAN interface) or can be given optional LAN information so the DHCP class id is only expected to be used with one particular LAN interface, for example, `dhcp_class_id[0/0/0]="Ignite_clients"`.

  This item is extremely useful if you have no control over the other DHCP servers on the network and do not want to accept temporary IP addresses from those servers. For example, the following configuration splits Itanium®-based and PA-RISC clients into two different DHCP classes:

```
is_hppa {
    dhcp_class_id = "Ignite_PA_clients"
} else {
    dhcp_class_id = "Ignite_IA_clients"
}
```

  The `else` condition implicitly means Itanium®-based systems since Ignite-UX currently supports only PA-RISC and Itanium®-based systems. This also assumes that you would want to share the same installation file system configuration. This condition effectively limits what the client responds to[52].

52 In a recovery situation, if you want to use DHCP to gain an IP address, Ignite-UX only adds the specific client IP address that actually created the recovery archive into the /etc/exports file. If you can accept DHCP responses from anyone, the potential

- `dhcp_server`—this keyword can be used to limit the responses that are accepted by a particular server.

  This is not as flexible as the `dhcp_class_id` keyword. If the DHCP server is down, no response is accepted. Compare that to a setup where you have two DHCP servers, each one willing to serve out different IP addresses to the one class id. If one Ignite-UX server goes down, the other DHCP server can still provide an IP address (although not as many are available).

  The `dhcp_server` configuration item can apply to all LAN interfaces (for example, `dhcp_server = "10.0.0.53"`) or to one particular LAN interface (for example, `dhcp_server[0/0/0] = "10.0.0.54"`).

- `dhcp_misc_opts`—the `dhcp_misc_opts` configuration item can apply to all LAN interfaces (for example, `dhcp_misc_opts = "-l 6"`) or to one particular LAN interface (for example, `dhcp_misc_opts[0/0/0] = "-l 6"`).

  Care should be taken when using the `dhcp_misc_opts` configuration item. The options provided are for `/usr/lbin/dhcpclient`. Since this is a back-end command, its options are not officially documented and this option should only be used when requested by HP.

  The only option you are likely to be asked to add is the option to print debugging output: `dhcp_misc_opts = "-l 6"`.

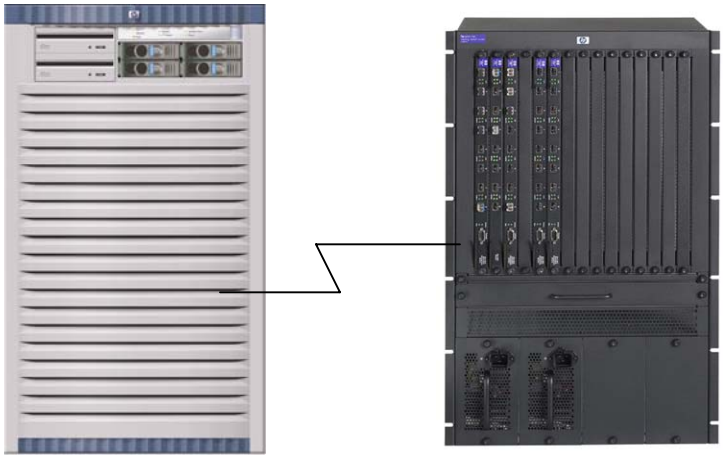
- `disable_dhcp`—when set to `TRUE` in the installation file system, this item prevents Ignite-UX from attempting to get an IP address using DHCP. You can use this item when you want to manually give Ignite-UX its information (or if you have defined the IP address information in the installation file system).

Problems that can be solved with **_hp_lanadmin_args**

When a kernel first boots (this is true even of a normal kernel that is not used for installation) all of the 100bt and Gigabit LAN interfaces attempt to autonegotiate the LAN speed and duplex settings.

In the example set up, an rp8400 is connected to an HP Procurve 9315m routing switch.

When the switch point to which port the system is connected is set to autonegotiate, the switch, and the system both autonegotiate the speed and duplex. When the switch is set to a speed and duplex value, autonegotiation fails



list of IP addresses that you may need to configure can get quite large, so many manual changes maybe required to the /etc/exports file on the Ignite-UX server. This assumes that archives are written to the Ignite-UX server.

between the system and the switch. This causes the system to revert to 100HD (half-duplex) and the switch continues using its hard-set speed (100FD or full-duplex, for example). Duplex mismatches that can occur in these circumstances can cause extreme throughput problems and Ignite-UX can appear hung or take a significantly longer time to complete tasks.

During a normal system startup, the system does not experience these problems because the system startup scripts have ideally been modified to set the speed and duplex values correctly for the LAN interface, assuming it is not going to autonegotiate. When using Ignite-UX, startup scripts cannot be used to set the speed and duplex values for your LAN interfaces.

To match the speed and duplex values, you must use the `_hp_lanadmin_args` variable to set these values correctly in the installation file system. This variable should *only* be set using `instl_adm`. For example, you could add the following configuration:

```
 ( lan[].driver == "btlan" )
{
_hp_lanadmin_args="-X 100FD"
}
```

This sets all LAN interfaces controlled by the LAN driver `btlan` to 100 full duplex in the installation file system by obtaining the current settings from the installation file system using the following command:

```
# instl_adm -d > /tmp/installfs.config
```

Next, you update the file to contain the extra configuration. If a similar configuration exists already, you may need to consider it when you add a new configuration. The updated configuration must be placed back into the installation file system using the following command:

```
# instl_adm -f /tmp/installfs.config
```

The `instl_adm` command attempts to keep the installation file systems synchronized. If you want only to apply a configuration to specific file systems, do not use the `-f` option; only use the `-F` option to give explicit installation file system names. However, you should try to keep the various installation file systems consistent.

This issue creates difficulties when you mix system types that are running at different duplexes and speeds, rather than setting the systems to autonegotiate both. To avoid this situation, you should develop and follow a consistent approach to speed and duplex settings.

If you have a gigabit switch set to only operate at 1000FD (gigabit speed with full duplex) and the gigabit interface on system is attempting to autonegotiate the speed and duplex, a 1000FD link will be achieved. This is because the only duplex supported at

gigabit speeds is full duplex. However, this is only true of 1000Base-T interfaces so 1000Base-SX interfaces must autonegotiate or they will fail to form a link.

Note that if you are using 1000Base-T interfaces and the switch is autonegotiating link speed and duplex or it is hard set to use 1000FD, you should not need to use `_hp_lanadmin_args` to force any speed/duplex values.

Control

The `_hp_keyboard` (see the "`_hp_keyboard`" section) control configuration item although not strictly an installation file system variable, it is a good idea to set this to a keyboard type if all of the systems that have keyboards are the same type. This prevents HP-UX from asking for a keyboard type when you have a graphics console.

The following control options are best set within an installation file system because Ignite-UX must be aware of them before it contacts either the Ignite-UX server or when it is attempting to start networking:

- `run_ui`—this configuration item controls whether the installation process should be interactive or not. If set to `FALSE`, the Ignite-UX GUI does not run and it is assumed that the configuration files provide the complete configuration required to install the system. If this is not `true` and there is not enough information, the installation changes to become interactive.

  Be careful when setting this item to `FALSE` as it is possible (if the system is booted from the network) to reinstall the system without any way to prevent it (except using the console, powering off, or resetting the system). This is regardless of how you set `control_from_server` (see next configuration item).

- `control_from_server`—this configuration item is used to determine if the Ignite-UX server that Ignite-UX has connected to should control the session or not. If set to `TRUE`, you must run `ignite` (the command) on the server and then manage the client remotely (you can break out of server control on the client console).
- `use_expert_ui`—this configuration item is very useful if you are an advanced user of Ignite-UX and want to run the Ignite-UX GUI most of the time instead of using "Wizard" mode. Using the Ignite-GUI, provides a much finer level of control over the system to be installed than Wizard mode does. To make the Ignite-UX GUI the default, add the following line to the installation file system:

  ```
  use_expert_ui = true
  ```

Environment variables

With the `env_vars` configuration item, you can place environment variables into the environment of the programs that are run by Ignite-UX. Setting an environment variable is done in the following way:

```
env_vars += "TZ=JST-9"
```

The preceding example sets the time zone for the installation to Japanese Standard Time.

Although this time zone does not define daylight savings, if you set a time zone that does have daylight savings you should be very careful. Take, for example, the time zone Australian Eastern time zone (EST-10EDT). Since the installation file system does not have a `tztab` file, when the time zone US EASTERN (EST5EDT) is in daylight savings it assumes EST-10EDT is also in daylight savings. In reality these time zones never actually overlap because one is in the northern hemisphere and the other in the southern.

The following environment variables are special and affect the installation or recovery process:

- `INST_NET_RESPONSE_TIMEOUT`—this variable provides safety for systems. The *`instl_adm`*`(4)` manpage states the following information about the variable:

  ```
  When booting an install client from the network, this sets the amount of
  time that the system will wait for a user response before it reboots in
  assumption that the system booted from the install server accidentally.
  Setting seconds to 0 (zero) will disable the timeout and the system will
  not prompt for a response.
  ```

  If this value is set to >0 the following message appears:

  ```
  Please press <return/enter> (within %d seconds) to continue loading the
  network-install utility:
  ```

  If that times out, it is followed by the following message, so users know that the system is rebooting:

  ```
  Unless you select <return/enter> within 10 seconds, the system will
  reboot:
  ```

If neither `Enter` nor `Return` is selected within 10 seconds, the system reboots.

- `INST_ALLOW_WARNINGS`—this variable again provides for the safety of systems (or in this case, disks):

  ```
  Setting this environment variable is useful for non-interactive install
  sessions when warnings about disks containing data cause the installation to
  switch to interactive mode.  Setting seconds to 1 will cause all warnings to
  be ignored and the installation will proceed. Setting seconds to greater than
  1 will allow the user that many seconds to read the warning and stop the
  installation by pressing <return>.
  ```

  You should be very careful about setting this variable away from the default of 0. A non-default value can be dangerous when you are doing a clean installation or cloning a system (especially to one with the same model string[53]). The following happens, depending on what value you set it to:

[53] When you set `INST_ALLOW_WARNINGS` to a non-zero value and you are performing a non-interactive recovery (especially cloning to the same hardware) or installation, it is possible that the warnings can be extremely important. For example, if you are cloning between systems of the same model but the boot disks are at different hardware paths (especially on a Fibre Channel attached array) and you do not interrupt the boot process in time, the installation/recovery session may have overwritten production data or data belonging to a different system.

0 – A yes/no question is asked: `Do you wish to cancel the non-interactive installation in order to respond to the warnings above?`. There is no timeout associated with this question; Ignite-UX waits indefinitely for a response.

1 – This message appears: `Continuing despite above warnings because INST_ALLOW_WARNINGS=1`. The installation/recovery proceeds regardless.

>1 – This message appears: `Press <Return/Enter> within <num> seconds to cancel batch-mode installation:`. You have that number of seconds to press **Return** or **Enter** to cancel the non-interactive session to enter the user interface.

- `INST_ENABLE_NETWORK`–this environment variable is useful when you have created media from which to boot a system, but you need the installation process to start networking (for example, one of the scripts used needs to contact another host). For example, setting this variable to 1 does this:

  ```
  env_vars += "INST_ENABLE_NETWORK=1"
  ```

  Unless `use_dhcp` has been set to `FALSE`, the system attempts to contact a DHCP server at that point.

- `LOADFILE_RETRY_COUNT`–this has the following description in the *`instl_adm`*`(4)` manpage:

  ```
  This can be used change the default number of times that the internal
  loadfile command will retry a failed attempt to retrieve data from the
  server or media.  Usually this retry mechanism is used to overcome tftp
  transfer problems.  The default value is 5.
  ```

As an alternative, you can use the `_hp_tftp_cmds` to change the re-transmission and timeout values when `tftp` is actually used.

Managing configurations with **`unifdef`**

You may find when writing configuration files that many of them tend to end up looking the same. There are at least two programs that you can use in conjunction with configuration files to make similar files more maintainable.

The program you can use at is `unifdef` (The second program is `m4`, but how to use `m4` is not discussed.)

The following example looks at the configuration files discussed in "Configuration examples – Part B". Suppose you have one very long configuration file that you want to save into an LIF, but over time you have finally exceeded the 8KB limit on configuration data so you have a configuration file that looks like the following:

```
server="10.0.0.2"
netmask[]="255.255.255.0"
route_gateway[0]="10.0.1.1"
route_destination[0]="default"
disable_dhcp=true

( lla[0/0/0/0] == "080009654321" ) {
        ip_addr[0/0/0/0]="10.0.1.45"
```

```
}
( lla[2/0/2] == "080009654123" ) {
        ip_addr[2/0/2]="10.0.1.46"
}
( lla[1/0/0/0] == "080009654213" ) {
        ip_addr[1/0/0/0]="10.0.1.47"
}
...

( lla[0/0/0/0] == "08000A654321" ) {
        ip_addr[0/0/0/0]="10.0.2.45"
}
( lla[2/0/2] == "08000A654123" ) {
        ip_addr[2/0/2]="10.0.2.46"
}
( lla[1/0/0/0] == "08000A654213" ) {
        ip_addr[1/0/0/0]="10.0.2.47"
}
```

You will want to use `unifdef` on this file so that you can maintain it in one file but also be able to create separate files to apply to LIFs (because you want to be able to set IP addresses automatically on the systems you boot). First, though, you need to insert some other things into the file.

The `unifdef` command recognizes the following C preprocessor directives:

```
#ifdef
#ifndef
#if
#else
#endif
```

With these directives, you can change the configuration file as follows:

```
server="10.0.0.2"
netmask[]="255.255.255.0"
route_gateway[0]="10.0.1.1"
route_destination[0]="default"
disable_dhcp=true

#ifdef LAN_SEGMENT_1
( lla[0/0/0/0] == "080009654321" ) {
        ip_addr[0/0/0/0]="10.0.1.45"
}
( lla[2/0/2] == "080009654123" ) {
```

```
        ip_addr[2/0/2]="10.0.1.46"
}
( lla[1/0/0/0] == "080009654213" ) {
        ip_addr[1/0/0/0]="10.0.1.47"
}
...
#endif
#ifdef LAN_SEGMENT_2
( lla[0/0/0/0] == "08000A654321" ) {
        ip_addr[0/0/0/0]="10.0.2.45"
}
( lla[2/0/2] == "08000A654123" ) {
        ip_addr[2/0/2]="10.0.2.46"
}
( lla[1/0/0/0] == "08000A654213" ) {
        ip_addr[1/0/0/0]="10.0.2.47"
}
...
#endif
```

You can now use `unifdef` to create a configuration file for LAN segment one:

```
$ unifdef -t[54] -DLAN_SEGMENT_1 -ULAN_SEGMENT_2 lan.cfg
server="10.0.0.2"
netmask[]="255.255.255.0"
route_gateway[0]="10.0.1.1"
route_destination[0]="default"
disable_dhcp=true

( lla[0/0/0/0] == "080009654321" ) {
        ip_addr[0/0/0/0]="10.0.1.45"
}
( lla[2/0/2] == "080009654123" ) {
        ip_addr[2/0/2]="10.0.1.46"
}
( lla[1/0/0/0] == "080009654213" ) {
        ip_addr[1/0/0/0]="10.0.1.47"
```

[54] The `-t` option treats the input as plain text rather than C style source code. Because this configuration does not look like C style source code, always use `the -t` option.

```
}
...
```

Note that `unifdef` is not a general C preprocessor; you must explicitly define or undefine variables before they are evaluated in the input file. If you require more functionality than `unifdef` provides, refer to *m4*(1).

Coping with **auto_adm** and boot changes in HP-UX B.11.23

Because PA-RISC systems have two possible kernels that they may need to boot with the release of Ignite-UX C.6.0, there needs to be some way to interact with the boot loader to boot the desired kernel.

Two criteria must be met:

- The installation process must be capable of a non-interactive installation or recovery.
- The installation process must provide a mechanism for selecting a kernel version.

These changes were made to the initial system loader (ISL) to enable it to understand a new format of `AUTO` file (this ISL is provided as part of Ignite-UX in the LIF file `/opt/ignite/boot/boot_lif`). This new format is capable of showing you a menu-like selection and asking you to choose a kernel to boot. Of course, to provide for non-interactive installation and recovery, there is a time-out value for a response and a default response.

With PA-RISC systems, using Ignite-UX version C.6.0.*x* or later, an HP-UX B.11.11 kernel is used to install and recover HP-UX B.11.11 and HP-UX 11.0 systems, and an HP-UX B.11.23 kernel is used to install and recover HP-UX B.11.23 systems. You cannot install or recover HP-UX B.11.11 or HP-UX 11.0 systems using the HP-UX B.11.23 kernel, and the HP-UX B.11.11 kernel cannot install or recover an HP-UX B.11.23 system. There is only a 64-bit HP-UX B.11.23 PA-RISC kernel. Ignite-UX prevents you from performing an unsupported installation using an incorrect kernel during installation.

Note the following:

- Currently, `auto_adm` only affects PA-RISC systems. There is only one Itanium®-based kernel shipped with Ignite-UX; no method is required to select between it and another kernel.
- This command exists from Ignite-UX version C.6.0.*x* onwards.
- With Ignite-UX version C.6.0.*x*, the new `auto_adm` formats in the `AUTO` file provide an enhancement to the current form of the `AUTO` file; they are not a replacement.

Looking at **auto_adm**

As stated in the *auto_adm*(1M) manpage, the following forms are allowed:

```
        /opt/ignite/bin/auto_adm -n -T timeout -M message -l label -c command
-b
                device -i image [-o outfile] [-O output_format] [-?]
```

```
/opt/ignite/bin/auto_adm -a -l label -c command -b device -i image
     [-f infile] [-o outfile] [-O output_format] [-?]

/opt/ignite/bin/auto_adm -A appendfile -f infile [-o outfile]
     [-O output_format] [-?]

/opt/ignite/bin/auto_adm -L label record_modifier [-f infile]
     [-o outfile] [-O output_format] [-?]

/opt/ignite/bin/auto_adm -d -L label [-f infile] [-o outfile]
     [-O output_format] [-?]

/opt/ignite/bin/auto_adm -D -L label [-f infile] [-o outfile]
     [-O output_format] [-?]

/opt/ignite/bin/auto_adm -T timeout [-f infile] [-o outfile]
     [-O output_format] [-?]

/opt/ignite/bin/auto_adm -M message [-f infile] [-o outfile]
     [-O output_format] [-?]
```

ISL and CONF format data

CONF data

The `auto_adm` manpage does not correctly document all of the allowed combinations of options. For instance, the preceding forms of the command do not allow the following example:

```
# auto_adm -f /opt/ignite/boot/boot_lif
timeout = 0
default = 1
message = Choose a boot action

label = target OS is B.11.00
bootcmd = boot
boot = (;0)
image = /boot/Rel_B.11.00/INSTALL
```

```
label = target OS is B.11.11
bootcmd = boot
boot = (;0)
image = /boot/Rel_B.11.11/INSTALL

label = target OS is B.11.23 PA
bootcmd = boot
boot = (;0)
image = /boot/Rel_B.11.23/WINSTALL
```

This reads the AUTO file from the LIF and prints it in the auto_conf (CONF) format.

In the CONF format, the timeout line specifies the amount of time (in seconds) to wait before taking the default action. The default line specifies the default action to take, which, in this case, boots a kernel for HP-UX 11.0. The message line specifies the message that is displayed after booting and showing the menu options to select a target operating system to boot.

The *auto_adm*(4) manpage lists the following definitions for label, bootcmd, boot, and image:

label=string

A string label that is displayed to the user on the console describing what action is performed if this boot image is loaded. For example, "Install 64-bit HP-UX B.11.11". The string need not be enclosed in quotes.

bootcmd=cmd

The SSL command to be executed if this image is selected. Refer to *hpux*(1M) for more information.

boot=hwpath

The hardware path relative to the boot LIF containing the desired boot image. Typically (0;). See *hpux*(1M) for more information.

image=path

The path of the image to be booted relative to the hardware path specified in the boot directive.

The bootcmd field should match the following form of the secondary loader (see *hpux*(1M)):

```
hpux [-F] [-lm] [-vm] [-lq] [-a[C|R|S|D] devicefile] [-fnumber]
 [-istring] [boot] [devicefile]
```

If, for example, you wanted to enable debug-level logging from boot logging automatically, you need to change the bootcmd string to the following:

```
bootcmd = -i3
```

The auto_adm command is intended for the management of the AUTO file in an LIF that is used by Ignite-UX. The bootcmd field only accepts one argument.

ISL data

The following example presents the same data as the CONF data although it is translated into ISL format:

```
# auto_adm -f ./output -o ./isl -O ISL
# cat isl
hpux KernelPrompt "Choose a boot action" 0 1
reset
"target OS is B.11.00" boot (;0)/boot/Rel_B.11.00/INSTALL
"target OS is B.11.11" boot (;0)/boot/Rel_B.11.11/INSTALL
"target OS is B.11.23 PA" boot (;0)/boot/Rel_B.11.23/WINSTALL
"Exit" reboot

# auto_adm -f ./lif -D -L "target OS is B.11.11" -O ISL
hpux KernelPrompt "Choose a boot action" 0 2
reset
"target OS is B.11.00" boot (;0)/boot/Rel_B.11.00/INSTALL
"target OS is B.11.11" boot (;0)/boot/Rel_B.11.11/INSTALL
"target OS is B.11.23 PA" boot (;0)/boot/Rel_B.11.23/WINSTALL
"Exit" reboot
```

The auto_adm command operates on CONF or ISL format data equally well. The default destination, if –o is not given, is to send the result to stdout.

Usage examples

Creating new files

This is the form of the auto_adm command used to create new files:

```
/opt/ignite/bin/auto_adm -n -T timeout -M message -l label -c command -b
  device -i image [-o outfile] [-O output_format] [-?]
```

The –n means that a new file is being created and the other options allow you to provide enough data to give the higher-level menu information and the first option on the menu. The following example creates the file:

```
# auto_adm -n -T 10 -M "Please select something to boot" \
```

```
> -l "HP-UX 11.0 32bit[55]" -c boot -b "(;0)" \
> -i boot/Rel_B.11.00/INSTALL -o boot.adm -O CONF
$ cat boot.adm
timeout = 10
default = 1
message = Please select something to boot

label = HP-UX 11.0 32bit[56]
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.00/INSTALL
```

To create the same thing but on ISL format, you can use the following format of the command:

```
# auto_adm -n -T 10 -M "Please select something to boot" \
> -l "HP-UX 11.0 32bit" -c boot -b "(;0)" \
> -i boot/Rel_B.11.00/INSTALL -o boot.isl -O ISL
$ cat boot.isl
hpux KernelPrompt "Please select something to boot" 10 1
reset
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"Exit" reboot
```

Adding new menu entries to a file

The following the form of the `auto_adm` command is used to add new menu selections into the file:

```
/opt/ignite/bin/auto_adm -a -l label -c command -b device -i image
   [-f infile] [-o outfile] [-O output_format] [-?]
```

First, add some entries into the `auto_conf` file in ISL format using `auto_adm`:

```
# auto_adm -a -l "HP-UX 11.0 32bit w/debug logging" -c "-i3" \
> -b "(;0)" -i boot/Rel_B.11.00/INSTALL -f boot.isl -o boot.isl2 -O ISL
```

---

[55] This format is not strictly 32-bit, as it correctly boots 32-bit and 64-bit capable systems (but not V class systems).
[56] Even though this menu choice says 32-bit, the PA-RISC primary loader (ISL) changes the `INSTALL` kernel to `WINSTALL` if the system can only run a 64-bit kernel. So, even though the one line says 32-bit, it actually supports both 32-bit and 64-bit systems.

```
# auto_adm -a -l "HP-UX 11.0 64bit" -c "boot" \
> -b "(;0)" -i boot/Rel_B.11.00/INSTALL -f boot.isl2 -o boot.isl -O ISL
$ cat boot.isl
hpux KernelPrompt "Please select something to boot" 10 1
reset
"HP-UX 11.0 64bit" boot (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.0 32bit w/debug logging" -i3 (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"Exit" reboot
```

You can make some similar changes in CONF format:

```
$ auto_adm -a -l "HP-UX 11.11 64bit" -c "boot" \
> -b "(;0)" -i boot/Rel_B.11.11/WINSTALL -f boot.adm -o boot.adm2 -O CONF
$ auto_adm -a -l "HP-UX 11.23 64bit" -c "boot" \
> -b "(;0)" -i boot/Rel_B.11.23/WINSTALL -f boot.adm2 -o boot.adm -O CONF
$ cat boot.adm
timeout = 10
default = 1
message = Please select something to boot

label = HP-UX 11.23 64bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.23/WINSTALL

label = HP-UX 11.11 64bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.11/WINSTALL

label = HP-UX 11.0 32bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.00/INSTALL
```

Using an "append" file

The auto_adm command also enables you to take records sitting in one file and append them to an input file, combining them into one output file. The following form does this:

```
/opt/ignite/bin/auto_adm -A appendfile -f infile [-o outfile]
   [-O output_format] [-?]
```

For example:

```
# cat boot.append
label = HP-UX 11.11 64bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.11/WINSTALL

label = HP-UX 11.11 32bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.11/INSTALL
# cat boot.isl
hpux KernelPrompt "Please select something to boot" 30 1
reset
"HP-UX 11.0 32bit w/debug logging" -i3 (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"Exit" reboot
# auto_adm -A boot.append -f boot.isl -o boot.combined -O ISL
# cat boot.combined
hpux KernelPrompt "Please select something to boot" 30 1
reset
"HP-UX 11.0 32bit w/debug logging" -i3 (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.11 64bit" boot (;0)boot/Rel_B.11.11/WINSTALL
"HP-UX 11.11 32bit" boot (;0)boot/Rel_B.11.11/INSTALL
"Exit" reboot
```

Note that the file given with the `-A` option must be in `CONF` format; if you use ISL format, the command generates errors.

Updating a menu entry in a file

The following form of `auto_adm` enables you to update a boot menu entry:

```
/opt/ignite/bin/auto_adm -L label record_modifier [-f infile]
   [-o outfile] [-O output_format] [-?]
```

The record modifier consists of the following:

record_modifier consists of at least one (and optionally any combination) of -l label, -c command, -b device, and/or -i image.

This enables you to update a boot menu entry. The following examples only look at CONF format files. First change the boot field for the label HP-UX 11.23 64bit:

```
# auto_adm -L "HP-UX 11.23 64bit" -c "-i3" \
> -f boot.adm -o boot.adm2 -O CONF
$ cat boot.adm2
timeout = 10
default = 1
message = Please select something to boot

label = HP-UX 11.23 64bit
bootcmd = boot
boot = -i3
image = boot/Rel_B.11.23/WINSTALL

label = HP-UX 11.11 64bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.11/WINSTALL

label = HP-UX 11.0 32bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.00/INSTALL
```

Rename the label HP-UX 11.23 64bit to HP-UX 11.23 64bit (PA-RISC):

```
$ auto_adm -L "HP-UX 11.23 64bit" -l "HP-UX 11.23 64bit (PA-RISC)" \
> -f boot.adm2 -o boot.adm -O CONF
$ cat boot.adm
timeout = 10
default = 1
message = Please select something to boot
```

```
label = HP-UX 11.23 64bit (PA-RISC)
bootcmd = -i3
boot = (;0)
image = boot/Rel_B.11.23/WINSTALL

label = HP-UX 11.11 64bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.11/WINSTALL

label = HP-UX 11.0 32bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.00/INSTALL
```

Deleting a menu entry from a file

The boot menu entries are removed from a file using the following form of `auto_adm`:

```
/opt/ignite/bin/auto_adm -d -L label [-f infile] [-o outfile]
   [-O output_format] [-?]
```

To remove a boot menu entry, you simply need to provide the label name from the file:

```
$ auto_adm -L "HP-UX 11.23 64bit (PA-RISC)" -d -f boot.adm -o boot.adm2 -O
CONF
$ cat boot.adm2
timeout = 10
default = 1
message = Please select something to boot
label = HP-UX 11.11 64bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.11/WINSTALL

label = HP-UX 11.0 32bit
bootcmd = boot
boot = (;0)
image = boot/Rel_B.11.00/INSTALL
```

Changing the default menu choice

Using `auto_adm`, you can also change the default menu choice with the following form of the command:

```
/opt/ignite/bin/auto_adm -D -L label [-f infile] [-o outfile]
   [-O output_format] [-?]
```

The following example shows the change in default menu choices from the first to the second option:

```
$ cat boot.adm
hpux KernelPrompt "Please select something to boot" 10 1
reset
"HP-UX 11.23 64bit (PA-RISC)" -i3 (;0)boot/Rel_B.11.23/WINSTALL
"HP-UX 11.11 64bit" boot (;0)boot/Rel_B.11.11/WINSTALL
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"Exit" reboot
$ auto_adm -D -L "HP-UX 11.11 64bit" -f boot.adm -o boot.adm2 -O ISL
$ cat boot.adm2
hpux KernelPrompt "Please select something to boot" 10 2
reset
"HP-UX 11.23 64bit (PA-RISC)" -i3 (;0)boot/Rel_B.11.23/WINSTALL
"HP-UX 11.11 64bit" boot (;0)boot/Rel_B.11.11/WINSTALL
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"Exit" reboot
```

Changing the timeout

The timeout value is used by the boot loader to determine how long to wait for user input before executing the default menu choice (see the next section for more information about how to set the default menu choice).

```
/opt/ignite/bin/auto_adm -T timeout [-f infile] [-o outfile]
   [-O output_format] [-?]
```

In the following example, you can see the timeout change from 10 to 30 seconds (the second last field on the first line) for this file in ISL format:

```
$ cat boot.isl2
hpux KernelPrompt "Please select something to boot" 10 1
reset
"HP-UX 11.0 32bit w/debug logging" -i3 (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"Exit" reboot
$ auto_adm -T 30 -f boot.isl2 -o boot.isl -O ISL
$ cat boot.isl
hpux KernelPrompt "Please select something to boot" 30 1
reset
"HP-UX 11.0 32bit w/debug logging" -i3 (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"Exit" reboot
```

Updating the prompt message

The prompt message is what the boot loader prompts you with when you are asked to make a selection from the available options. The form of the command that enables you to change the prompt message is as follows:

```
/opt/ignite/bin/auto_adm -M message [-f infile] [-o outfile]
   [-O output_format] [-?]
```

The following example uses this form to change the prompt message:

```
# auto_adm -M "Please select an option" -f boot.combined -o boot.final -O ISL
# cat boot.final
hpux KernelPrompt "Please select an option" 30 1
reset
"HP-UX 11.0 32bit w/debug logging" -i3 (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.0 32bit" boot (;0)boot/Rel_B.11.00/INSTALL
"HP-UX 11.11 64bit" boot (;0)boot/Rel_B.11.11/WINSTALL
"HP-UX 11.11 32bit" boot (;0)boot/Rel_B.11.11/INSTALL
"Exit" reboot
```

Into and out of an LIF file

The `auto_adm` command automatically recognizes an LIF file when provided as the argument to `-f` or `-o` options and handles the format of the data correctly. In the following example, the data is extracted from the boot LIF provided with Ignite-UX and at

the same time changes the timeout from 120 seconds to 30 seconds. You can compare the data with `lifcp`:

```
$ auto_adm -T 30 -f ./boot_lif -o boot.tmp -O ISL
$ lifcp ./boot_lif:AUTO -
hpux KernelPrompt "Choose Operating System to Install :" 120 1
reset
"target OS is B.11.00" boot (;0)/boot/Rel_B.11.00/INSTALL
"target OS is B.11.11" boot (;0)/boot/Rel_B.11.11/INSTALL
"target OS is B.11.23 PA" boot (;0)/boot/Rel_B.11.23/WINSTALL
"Exit" reboot
$ cat boot.tmp
hpux KernelPrompt "Choose a boot action" 30 1
reset
"target OS is B.11.00" boot (;0)/boot/Rel_B.11.00/INSTALL
"target OS is B.11.11" boot (;0)/boot/Rel_B.11.11/INSTALL
"target OS is B.11.23 PA" boot (;0)/boot/Rel_B.11.23/WINSTALL
"Exit" reboot
```

You can put the data back into an LIF by specifying the LIF as the output file (it is placed into the `AUTO` file within the LIF).

Installation configurations using Software Distributor depots

The following is an example process for configuring support on an Ignite-UX server to allow you to install systems using Software Distributor (SD) depots as the repository for the software.

It is important to note the following:

- How to configure `instl_bootd` or DHCP servers to allow clients to boot over the network is not discussed. For more information, refer to the *Ignite-UX Administration Guide* and *`instl_bootd`*`(1M)`.
- How to setup depots up remote to the Ignite-UX server is not discussed; the simplest location for depots is on the Ignite-UX server.
- How to use this process with versions earlier than the C.6.0.*x* version of Ignite-UX is not discussed. If you are using a version of Ignite-UX and some commands do not work as shown, you should consider updating your Ignite-UX.

You should review the entire section before following any of the instructions because consequences and issues related to running commands are discussed throughout.

Getting started

The first step is to choose the HP-UX 11i v1 Operating Environment (OE) media that you want to use. To make an informed decision you should review the *Cloning and Recovery Issues using Ignite-UX* White Paper. The issues that this paper discusses are relevant to installing a system (particularly a new system) because you must choose the OE that provides support for all of the hardware in the system.

Creating the core operating system depot

To create the core operating system depot the *make_depots* command is used. You will be copying the December 2003 HP-UX 11i Version 1 OE media into a core operating system depot in this example. You will use the defaults for the `make_depots` command; you only need to give the block device[57] of the DVD drive to define the location of the final depot and the release you will be copying as follows:

```
make_depots -r B.11.11 -s /dev/dsk/c0t1d0
```

If the command is successful, no output is displayed by `make_depots`.

Important:
To successfully copy the contents of the core operating system DVD[58] to a depot on the Ignite-UX server, you *must* have the patches that enable the Rock Ridge extensions natively on HP-UX installed on the Ignite-UX server. Also, to successfully copy the contents of the HP-UX 11i Version 2 May 2005 or later OE media on an 11i Version 1 or 11.0 Ignite-UX server, you must upgrade to Ignite-UX version C.6.2 and have the patches that enable the Rock Ridge extensions natively on HP-UX installed on the Ignite-UX server. On an 11i Version 2 Ignite-UX server, you need only to upgrade to Ignite-UX version C.6.2.

The system on which this example was developed (running HP-UX B.11.11) required these patches:

```
PHCO_25841    1.0           Add Rock Ridge extension to mount_cdfs(1M)
PHKL_26269    1.0           Rock Ridge extension for ISO-9660
PHKL_28025    1.0           Rock Ridge extension for ISO-9660
```

There are no patches required for HP-UX B.11.23. The equivalent patches for HP-UX 11.0 are:

```
PHCO_26449    1.0           Add Rock Ridge extension to mount_cdfs(1M)
```

[57] The manpage for `make_depots` indicates that a DVD/CD-ROM drive must be given using the block device. A character device would be assumed to be a tape dive containing a serial core operating system depot on a tape. Also you can give a normal SD depot in the form `<host>:<path>`; a regular file name would be considered a tape depot on disk.
[58] HP-UX 11.0 is delivered on one CD only; only that CD is needed to install a system. HP-UX 11i v1 is delivered on both CD and DVD. HP-UX 11i v2 is only delivered on DVD.

```
PHKL_26450    1.0            Rock Ridge extension for ISO-9660
PHKL_28060    1.0            Y2k; Rock Ridge extension for ISO-9660
```

The `make_depots` command above assumes that there was no previous depot created at `/var/opt/ignite/depots/Rel_B.11.11/core`. If such a depot exists, you should use the `-d` option of `make_depots` to have the new depot created in a new location. If you need to maintain multiple core operating system (OE) depots for a release of HP-UX, you should consider placing them into depots that indicate what version of the media the depot contains for easy identification. For example, if you are creating depots for the December 2003 and June 2004 HP-UX 11i v1 Mission Critical OE media you could use the following commands:

```
make_depots -r B.11.11 -s /dev/dsk/c0t1d0 -d \
    /var/opt/ignite/depots/Rel_B.11.11/core_1203_mc
make_depots -r B.11.11 -s /dev/dsk/c0t1d0 -d \
    /var/opt/ignite/depots/Rel_B.11.11/core_0604_mc
```

Important:
If you need to install different OEs from your Ignite-UX server, you must ensure that the only the OEs that the client system is licensed to run are installed on the system.

Alternatively, you could use the following commands to remove the contents of the existing depot before running the previous commands:

```
swreg -u -l depot /var/opt/ignite/depots/Rel_B.11.11/core
rm -rf /var/opt/ignite/depots/Rel_B.11.11/core
```

If everything is successful, you will have a depot at the following location containing the OE media for HP-UX B.11.11:

```
swlist -d -s /var/opt/ignite/depots/Rel_B.11.11/core

# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.11/core
#

#
# Bundle(s):
#

  100BaseT-00        B.11.11.01     EISA 100BaseT;Supptd W=A4308B;SW=J2780BA

  100BaseT-01        B.11.11.01     HP-PB 100BaseT;Supptd HW=A3495A;SW=J2759BA
...
```

```
   HPUXBase32          B.11.11         HP-UX 32-bit Base OS
   HPUXBase64          B.11.11         HP-UX 64-bit Base OS
   HPUXBaseAux         B.11.11.0312    HP-UX Base OS Auxiliary
   HWEnable11i         B.11.11.0312.4 Hardware Enablement Patches for HP-UX 11i
v1, December 2003
...
   hpuxwsWebmin        A.1.0.05.02     HP-UX Webmin-based Admin
   hpuxwsXml           A.1.0.01.02     HP-UX XML Web Server Tools
   perl                B.5.6.1.F       Perl Programming Language
   scsiU320-00         B.11.11.00      PCI SCSI U320; Supptd HW=A7173A
```

The log file that *make_depots* creates contains more detailed information. By default, the log file is in `/var/opt/ignite/logs/make_depots`.

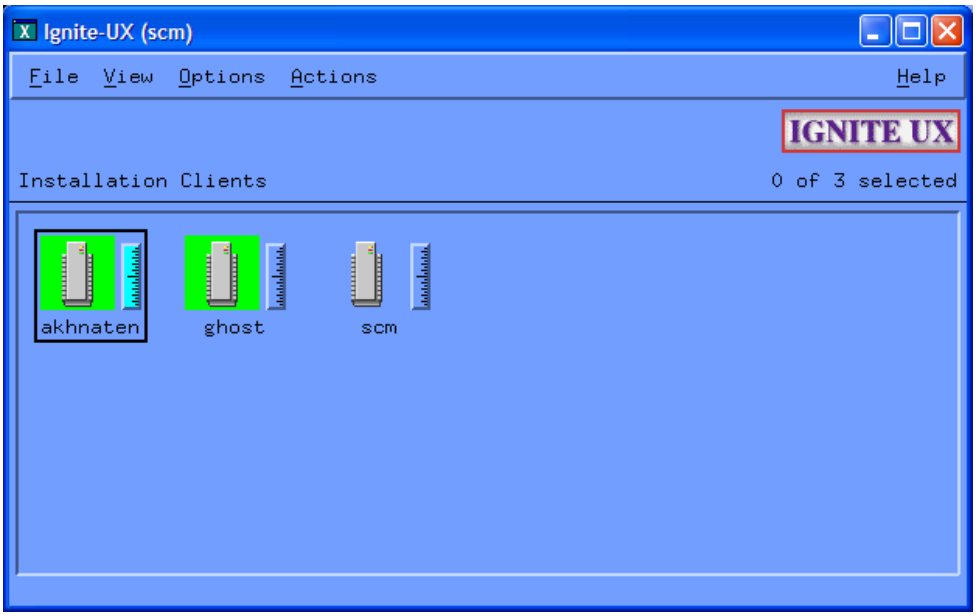
Other methods for creating the Core OS depot

There are other methods for creating the Core OS depot[59]. Some better than others and some having unique pitfalls.

Using the Ignite-UX GUI

You can use the Ignite-UX GUI on your Ignite-UX server to create a core operating system (OE) depot using the following process:

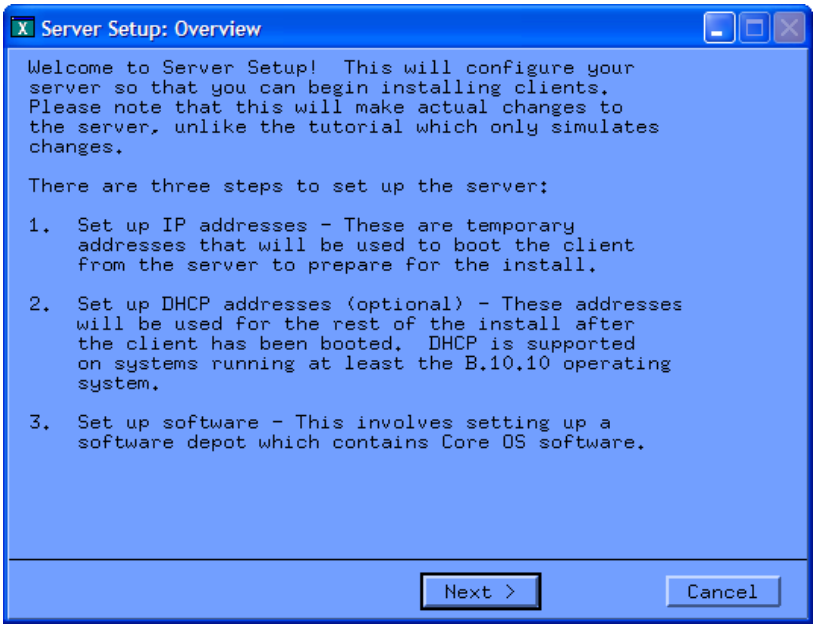
1. On your Ignite-UX server, enter **`/opt/ignite/bin/ignite`**.



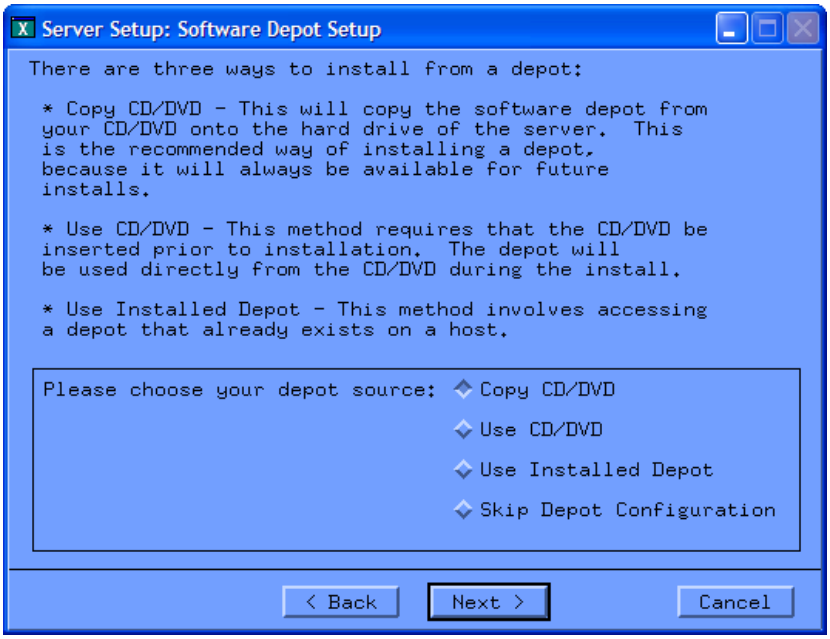
2. Select **Run Tutorial/Server Setup...** from the **Actions** menu.

[59] Note that `make_depots` and the option in the Ignite-UX GUI to setup software depots is not just for core operating system (OE) media, you can copy other media using this method such as the applications CD/DVD as well.

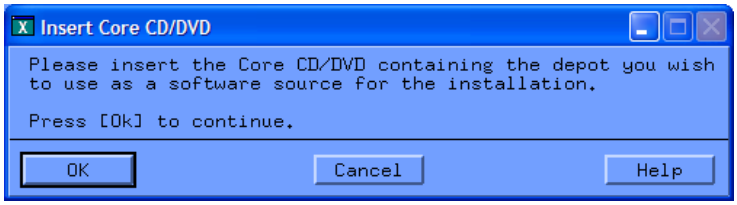
3. Click the **Server Setup** button.



4. The third step, **Set up Software**, allows you to create a core operating system depot. Click the **Next** button in this and the **Next** or **OK** buttons on the following dialog boxes until the following dialog box appears.



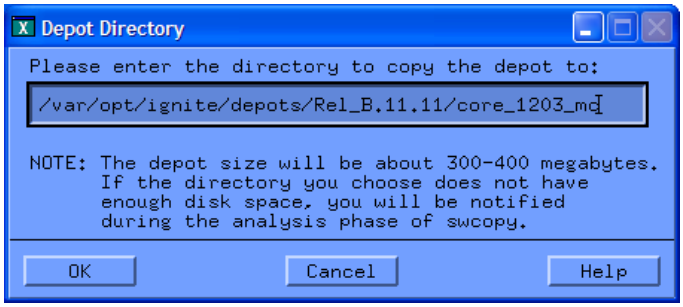
5. Select **Copy CD/DVD** using the adjacent radio button then click **Next** to copy the core operating system (OE) media into a depot. Note the **Use CD/DVD** option, which allows you use a directly mounted DVD rather than have it copied to another depot for usage. If you had multiple DVD drives attached to your Ignite-UX server, you could mount all of the depots directly from their DVDs and save disk space.[60]



6. Insert the media as requested and click **OK**.

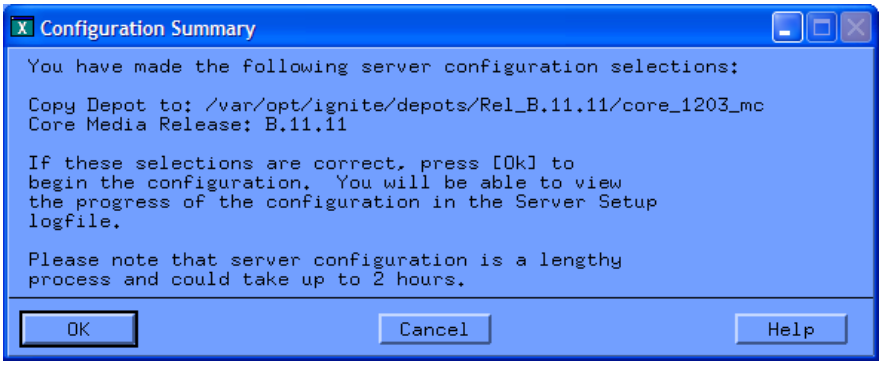
[60] If you manually mount a DVD, you can run `make_config` on the depot and then create a `cfg` clause referencing the configuration file. You *must* register the depot on the DVD using `swreg` to avoid errors. When remote systems attempt to access the mounted DVD as a depot they may be denied access to it. To determine whether a depot is registered or not, use **`swlist -l depot`**, which displays the path to the depot if it is registered.

7. Select the correct CD or DVD device from the drop-down button then click **OK**.

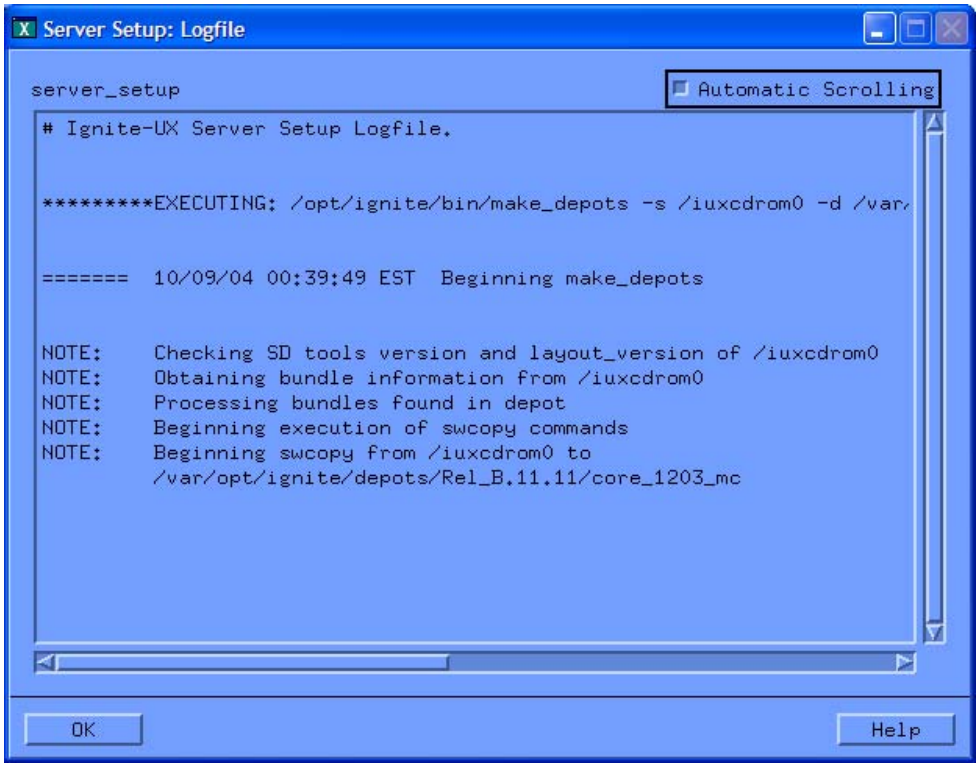


8. Enter the location and name of the depot that the core operating system (OE) to which the media will be copied then click **OK**.

The size displayed in the dialog box may not be correct as it is a static message and the 3-400Mb is not based upon the size of the depot on the CD or DVD.

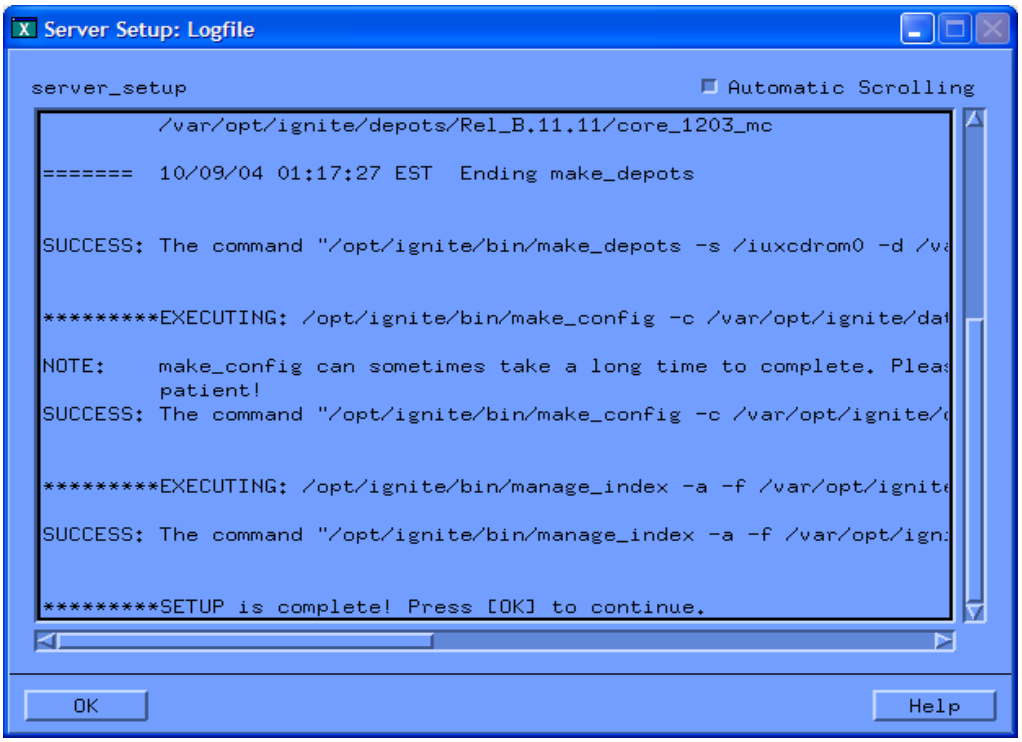


9. Review the information you have entered to be sure it is correct and then click **OK**. A status dialog box appears so that you can monitor the software copying.

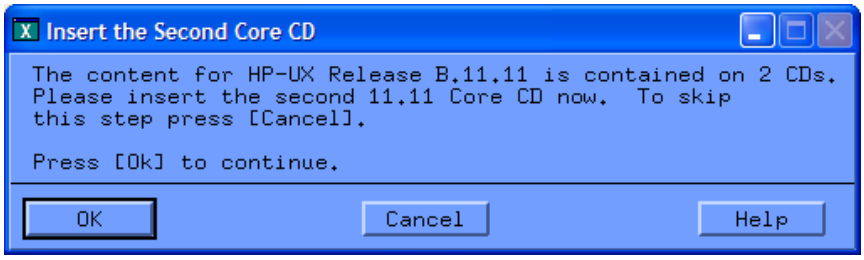


You should not click **OK** in the status dialog box until the process of copying the CD/DVD is complete as an error results.

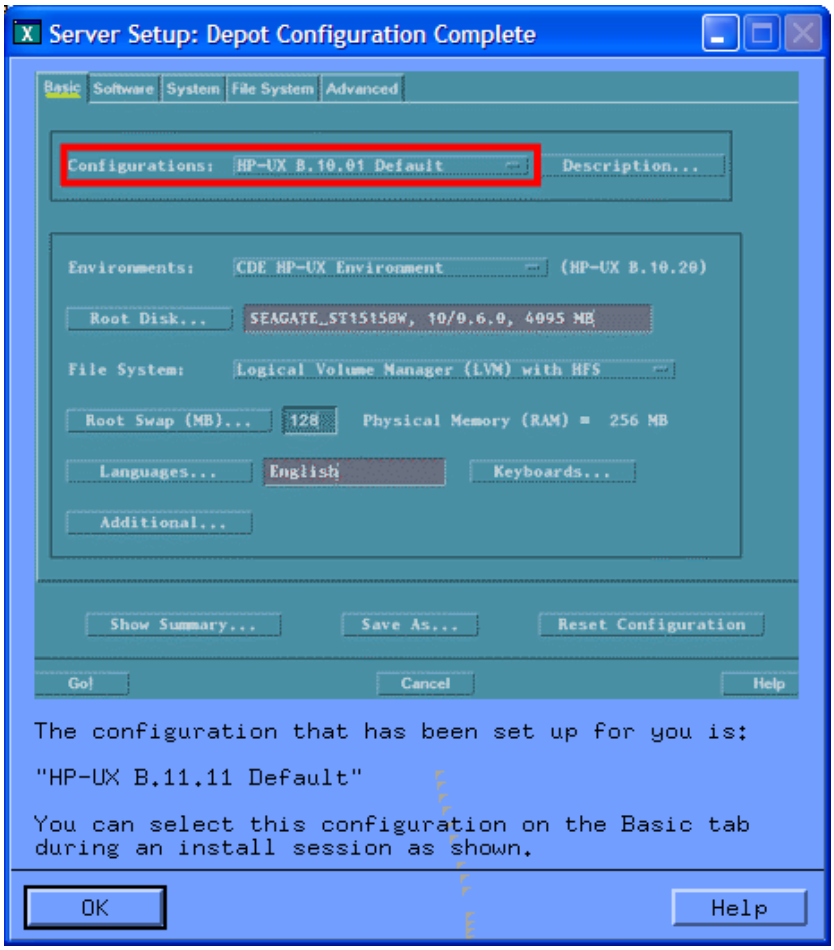
10. Once complete click **OK** and a log file dialog box similar to the following appears:



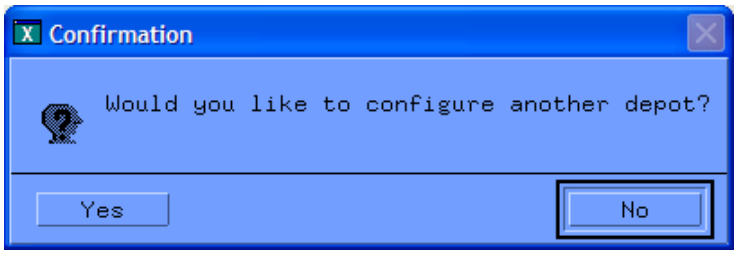
Once the process is completes successfully: the depot exists, a configuration file is created for it, and the configuration file is added to the appropriate default configuration clause `/var/opt/ignite/INDEX`. If you do not want the configuration file added to the `cfg` clauses in the `INDEX` file that it has been added to, you must remove it from each clause using `manage_index`. Allowing `manage_index` to update the `/var/opt/ignite/INDEX` file indiscriminately may have unintended side affects so you should review the `INDEX` file to ensure that it remains correct for your environment.



11. If you are using CDs, change the CD and click **OK**; if you are using a DVD click **Cancel** then **OK** in the next dialog box to continue. The following dialog box appears:



12. Click **OK**.



13. Click **Yes** to setup more depots or **No** to finish the process.

14. Click **Finish** to complete the process.

The pitfalls of using CDs instead of DVDs

If you are creating a depot, you should be very careful if you are using the OE CDs instead of the OE DVD. The `make_depots` command has a limitation in that it will not prompt for subsequent CDs and will only copy the first CD into the depot. However, it is safe to subsequently re-run the `make_depots` command on each of the CDs to copy them into the same depot; do *not* copy them into different depots.

Creating the configuration file to describe the depot

The `make_config` command is used to create the configuration file that describes the core operating system (OE) depot you just created. The easiest way to do this since the depot is local and in `/var/opt/ignite/depots/Rel_B.11.11` is as follows:

```
make_config -s /var/opt/ignite/depots/Rel_B.11.11/core \
    -c /var/opt/ignite/data/Rel_B.11.11/core_cfg
```

Warning:
HP does not support placing different versions of an OE in one depot. Instead, you must keep different OEs in different depots.

This creates the configuration file that describes the depot in `/var/opt/ignite/data/Rel_B.11.11` and it is named `core_cfg`[61]. However, it does not update `/var/opt/ignite/INDEX` so that you can create your own custom `cfg` clause later. If everything completes successfully, you should see the following message only:

```
NOTE:    make_config can sometimes take a long time to complete. Please be
         patient!
```

An alternative to this is to use the following command:

```
make_config -r B.11.11
```

This command automatically creates configuration files for all of the depots under `/var/opt/ignite/depots/Rel_B.11.11`. However, this form of the command will also update the `/var/opt/ignite/INDEX` file (typically updating all `cfg` clauses affecting the given HP-UX release with information about the configuration files). This may not be desirable.

For example, if you run the following command:

```
make_config -r B.11.11 -s core_1203_mc
```

Then the configuration file will be created and the `/var/opt/ignite/INDEX` file will be updated (only showing the Default B.11.11 clause) to be:

```
# cat /var/opt/ignite/INDEX
...
cfg "HP-UX B.11.11 Default" {
        description "This selection supplies the default system configuration
that HP supplies for the B.11.11 release."
        "/opt/ignite/data/Rel_B.11.11/config"
        "/opt/ignite/data/Rel_B.11.11/hw_patches_cfg"
```

[61] This is why it was suggested earlier that if you have multiple core operating system (OE) depots, you should give them names that include the release date of the media and an indication (if required) of the OE type.

```
        "/var/opt/ignite/data/Rel_B.11.11/core_1203_mc_cfg"
        "/var/opt/ignite/config.local"
}
```

If you then run the following command:

```
make_config -r B.11.11
```

In addition to creating the configuration file, the `/var/opt/ignite/INDEX` file will be updated as in the following excerpt of the file:

```
cfg "HP-UX B.11.11 Default" {
        description "This selection supplies the default system configuration
that HP supplies for the B.11.11 release."
        "/opt/ignite/data/Rel_B.11.11/config"
        "/opt/ignite/data/Rel_B.11.11/hw_patches_cfg"
        "/var/opt/ignite/data/Rel_B.11.11/core_1203_mc_cfg"
        "/var/opt/ignite/data/Rel_B.11.11/core_cfg"
        "/var/opt/ignite/config.local"
}
```

You should avoid multiple core operating system (OE) definitions within the one `cfg` clause. The contents of the second configuration file defining a core operating system (OE) depot override the configuration in the first.

This command does not cause any additional changes to occur and `make_config` would only add the `core_cfg` configuration file to every `B.11.11` clause in the `INDEX` file.

```
# make_config -r B.11.11 -s core
```

For information about the contents of core operating system (OE) configuration file, see the "Core operating system depot configuration" section.

Creating a minimalist **`cfg`** clause for installation

You can create a `cfg` clause in `/var/opt/ignite/INDEX` that will allow you to install a system.

```
manage_index -n "HP-UX B.11.11 Default" -c \
    "Custom B.11.11 installation using SD"
manage_index -a -f /var/opt/ignite/data/Rel_B.11.11/core_cfg \
    -c "Custom B.11.11 installation using SD"
```

You use `manage_index` to avoid using the variations of `make_config` that indiscriminately change `cfg` clauses in the `INDEX` file. In this case, you are copying the default `B.11.11` clause (the changes from `make_config` in the previous section have been undone prior to doing this step).

When complete you have a new `cfg` clause in the `INDEX` file:

```
# cat /var/opt/ignite/INDEX
...
cfg "Custom B.11.11 installation using SD" {
        description "This selection supplies the default system configuration
that HP supplies for the B.11.11 release."
        "/opt/ignite/data/Rel_B.11.11/config"
        "/opt/ignite/data/Rel_B.11.11/hw_patches_cfg"
        "/var/opt/ignite/data/Rel_B.11.11/core_cfg"
        "/var/opt/ignite/config.local"
}
```

Comments on the SD based **`cfg`** clause

The clause created does nothing except install a generic core operating system (OE) from an SD depot (with the patches included with the operating system). The important thing is that it will install HP-UX. To enable this all you have had to do is create a core operating system (OE) depot, run `make_config` once, and then `manage_index` twice.

In the following sections, you will add to this by adding new depots to allow additional applications to be installed and custom configuration to be executed.

You have saved a lot of time by using the default configuration files that Ignite-UX supplies for every HP-UX revision (the `config` and `hw_patches_cfg` files) as a basic configuration for file systems and some additional configuration for patching. The release-specific (in this case `/opt/ignite/data/Rel_B.11.11/config`) configuration file supplies some important variable definitions. Including this file prevents changes to Ignite-UX in the future from impacting you because you automatically receive changes to the release-specific `config` and `hw_patches_cfg` files as with each new release of Ignite-UX.

How to setup your environment to boot systems so that they can contact your Ignite-UX server to allow you to install systems is not discussed in this paper. For more information on completing the setup of your server, refer to the *Ignite-UX Administration Guide*.

Extending the generic SD **`cfg`** clause with applications

In this section, you will enhance the core operating system (OE) bundle that you previously created by making available some of the applications from the HP Application Media. In this example, you will be copying the December 2003 Application DVD. The application media is created and copied in much the same way as "Creating the core operating system depot".

Creating the application depot

First, you must copy the applications from the DVD into a depot so that you can use them. The following command begins the process:

```
make_depots -d /var/opt/ignite/depots/Rel_B.11.11/apps_1203 \
    -s /dev/dsk/c0t1d0
```

If the media contains codeword-protected applications, so you must supply the codewords directly to the `make_depots` command (for more information, see *`make_depots`*`(1M)`). This command can take a long time to complete.

Creating a configuration for the application depot

Since you created the depot, you can create the configuration file for the depot using the following command:

```
make_config -s /var/opt/ignite/depots/Rel_B.11.11/apps_1203 \
    -c /var/opt/ignite/data/Rel_B.11.11/apps_1203_cfg
```

Remember that you do not use the `-r` option so that `manage_index` is run by `make_config` to update every `cfg` clause that uses HP-UX B.11.11 to include the file. You should only add the `apps_1203_cfg` file to one `cfg` clause.

After you run the following command to add `apps_1203_cfg` to the `cfg` clause, you can install any applications that are in the depot:

```
manage_index -a -f /var/opt/ignite/data/Rel_B.11.11/apps_1203_cfg \
    -c "Custom B.11.11 installation using SD"
```

An example problem

It is often helpful to examine a real life problem. The previous process created a problem by adding the applications configuration file to the same `cfg` clause as the core operating system (OE) depot. This issue was touched on previously and this section describes it in further detail.

In `/var/opt/ignite/data/Rel_B.11.11/core_cfg` is the following definition of Perl:

```
init sw_sel "perl" {
    description = "Perl Programming Language"
    sw_source = "core"
    sw_category = "OrderedApps"
    sd_software_list = "perl,r=B.5.6.1.F,a=HP-UX_B.11.11_32/64,v=HP"
    impacts = "/opt" 68657Kb
} = TRUE
```

In `/var/opt/ignite/data/Rel_B.11.11/apps_1203_cfg` is this definition of Perl[62]:

```
init sw_sel "perl" {
    description = "Perl Programming Language"
    sw_source = "/var/opt/ignite/depots/Rel_B.11.11/apps_1203"
    sw_category = "OrderedApps"
```

[62] Perl is not the only product affected by the issue. There are other products that the core operating system (OE) media and the applications media have in common.

```
    sd_software_list = "perl,r=B.5.6.1.F,a=HP-UX_B.11.11_32/64,v=HP"
    impacts = "/opt" 68657Kb
} = TRUE
```

The two previous examples have the same `sw_sel` name so they are the same. When the second `perl sw_sel` clause is parsed, Ignite-UX assumes that it is an update to the existing definition of Perl already found because the clauses are named identically. The attributes of the `sw_sel`, such as the `sw_source`, are updated by the second definition, which replaces the first definition of Perl.

Perl is an always-loaded product; by default, it is marked to be loaded. There is the possibility that this could cause a problem if the application media and the core operating system (OE) media do not have the same release date. If you have newer core operating system (OE) media and a much older application depot being referenced, problems like the following could occur:

- An older version of Ignite-UX is referenced in the applications depot and it may not support the system being installed. For example, if you create a recovery tape and the installation kernel on the tape does not support the current system it may panic on first boot when you need to recover the system.
- The kernel drivers in the application depot are also located in the core operating system (OE) depot including SCSI, Fibre Channel, and Gigabit Ethernet drivers. If you are referencing older versions of the drivers and the system is newer, you may have hardware in the system that is not supported by the older drivers and this causes problems.

There are four solutions to this problem:

- Edit the `apps_1203_cfg` file and change the name of the `sw_sel` clauses so that they are unique and descriptive of the clause's content. This allows you to address the two software selection clauses uniquely. It also ensures that the intended software is selected from the core operating system (OE) depot, and that software from the applications media is not selected inadvertently. This is time consuming and if you want to update the configuration file using `make_config` you must reapply the changes.
- Remove all of the software bundles that exist in the core operating system (OE) depot and the applications depot. You would run `make_depots` on the applications DVD then use `swremove` on the applications depot. HP does not recommend this solution.
- Place the application media into the core operating system (OE) depot rather than putting it into a separate depot. This means that you must manage the depots as one. Note that you cannot update an OE in a depot. If you wish to use a new OE in an existing depot, you must remove the entire contents of the depot before using `make_depots` to copy in new software from OE and applications media.
- Ensure that the Core OS (OE) media and applications media are always synchronized.

Resolving applications that are not in SD format

There are two solutions when you find that you have an application that is not in SD format:

- Package an application in SD format
- Define an application in a non-SD format

The following topics describe both solutions.

Package an application in SD format

You could learn how to write an SD product specification file (psf) for use with `swpackage` to produce an SD depot of your application. Learning how to write a PSF can be difficult and time consuming. For information regarding the creation of PSF files, see *Software Distributor Administration Guide for HP-UX 11i*.

Alternatively, you can use the HP Software Package Builder (SPB) product makes it easy to package applications into SD format. SPB provides a window into the software package structure, showing attributes that can be set for each package element.

You can find SPB product information and download it from the HP Software Depot at:

http://software.hp.com/portal/swdepot/displayProductInfo.do?productNumber=SwPkgBuilder

Once you have the software in SD format, using either solution, you can:

1. copy it into a depot;
2. run `make_config` to generate a config file;
3. then run `manage_index` to add the configuration file to the desired `cfg` clause in `/var/opt/ignite/INDEX`.

Important:
SD depots must contain software in bundles after you have packaged the software. If it is not packaged at the bundle level, you will have to create a bundle for it with the `make_bundles` command.

Define an application in a non-SD format

You either have the option of providing applications in non-SD format in `tar` or `cpio` format archives that have been gzipped or compressed. This section explains how to write a configuration file that describes an application in non-SD format[63].

The archive that you will be installing the application with is assumed to contain no system specific configuration.

You should configure the application as necessary with the command and script hooks provided by Ignite-UX. The scripts that may be executed while installing the application should *never* query the user because not all installations are attended. For a non-interactive installation (the user does not have access to the console), a script that seeks input from the console would effectively hang the installation process.

---

[63] This section is not relevant to core operating system (OE) golden images. Additional configuration steps are required to define a core operating system (OE) versus an application golden image.

Writing the configuration file for an application in non-SD format

In the following example, the application configuration file that this configuration will be placed in is `/var/opt/ignite/data/Rel_B.11.11/myapps_cfg`.

```
#
# Definition of where my application is located
#
sw_source "my application archive" {
    description = "my application"
    source_format = archive
    source_type="NET"
    nfs_source = "10.2.72.150:/var/opt/ignite/My_Applications"
}
```

Next, you define a `sw_source` for the application including its location on the NFS server (at IP address 10.2.72.150) in the `/var/opt/ignite/My_Applications` directory. Remember, this directory must be exported from the NFS server so that it can be mounted. If you choose to limit access to this directory, remember that you may want servers that obtain IP address via DHCP to be able to mount the directory when setting its permissions.

```
sw_category "SiteApps" {
    description = "My site applications"
}
```

The previous example defines a category that you can easily pick out in the list of applications in the **Software** tab of the Ignite-UX GUI.

Next, define a 64-bit version of the application and ensure that you can load both the 32-bit and 64-bit versions of the application by making them exrequisties of each other. The impacts statements are generated with the `archive_impact` command, and then manually tailored to account for extra space in `/opt`, `/var`, and `/tmp` that the application requires as in the following example.

```
sw_sel "MyApp,v=1.0,64bit" {
    description = "My Application version 1.0"
    sw_source = "my application archive"
    sw_category = "SiteApps"
    archive_type = gzip tar
    archive_path = "myapps64-1.0.tar.gz"
    impacts = "/opt"   160154Kb
    impacts = "/tmp"      500Kb
    impacts = "/var"   450334Kb
    exrequisite="MyApp,v=1.0,32bit"
    post_load_cmd="/opt/myapp/bin/initialize"
}
```

You must ensure that only one version of the application is installed and that the user cannot change it. For example, allowing a user to install the 64-bit version with 32-bit HP-UX. The following example automatically selects the version based upon the `HP-UX Base OS` bundle has been selected.

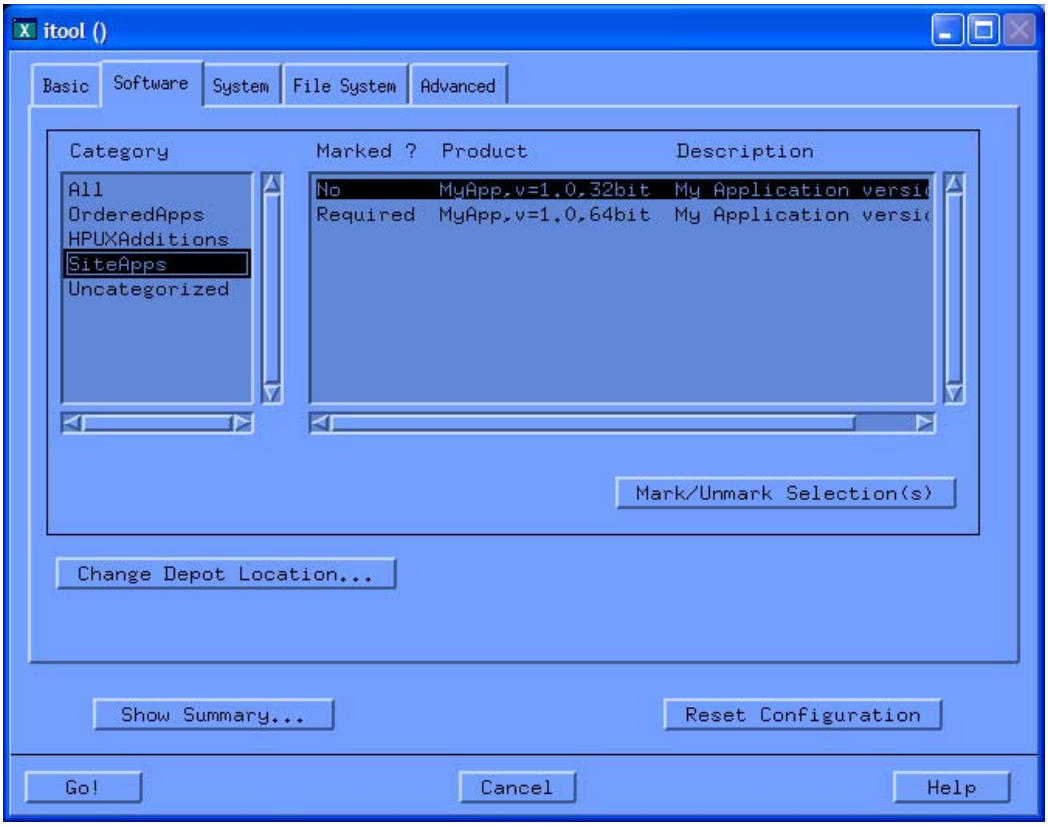
```
sw_sel "MyApp,v=1.0,32bit" {
    description = "My Application version 1.0"
    sw_source = "my application archive"
    sw_category = "SiteApps"
    archive_type = gzip tar
    archive_path = "myapps32-1.0.tar.gz"
    impacts = "/opt"   165154Kb
    impacts = "/tmp"      500Kb
    impacts = "/var"   450334Kb
    exrequisite="MyApp,v=1.0,64bit"
    post_load_cmd="/opt/myapp/bin/initialize"
}
```

The software selection of `MyApp` was not created using `init` so the user cannot change the selection. It cannot be deselected nor can the version be changed.

```
(sw_sel "HPUXBase64") {
    sw_sel "MyApp,v=1.0,64bit"=TRUE
}

(sw_sel "HPUXBase32") {
    sw_sel "MyApp,v=1.0,32bit"=TRUE
}
```

The new software selection you created, `MyApp`, appears on the **Software** tab of the Ignite-UX GUI:



The selection of the 64-bit product is marked as `Required` because the 64-bit OE has been selected.

If you placed the `init` before the `sw_sel`, the Ignite-UX GUI would allow you to change the software selection. The software would then be marked with Yes or No to indicate whether it will be installed. This is not recommended because it allows users to install the 64-bit version with a 32-bit installation; this should be avoided and is illustrated in the following example:

```
(sw_sel "HPUXBase64") {
    init sw_sel "MyApp,v=1.0,64bit"=TRUE
}

(sw_sel "HPUXBase32") {
    init sw_sel "MyApp,v=1.0,32bit"=TRUE
}
```

Since the examples presented are installing onto HP-UX B.11.11, you do not have to worry about issues resulting from code compiled for PA-RISC or Itanium-based systems. If this is a consideration as it is with HP-UX B.11.23, you must use the following test to determine the system type. For more information, see *instl_adm*(4).

```
is_hppa
```

```
            is_ia64
                 Boolean values that indicate if a system is PA-RISC or
                 Itanium®-based architecture, respectively.
```

Additional system hardware tests are explained in the following section.

Using **noncore.cfg** to define applications

Ignite-UX provides a template that you can use to define non-SD (archive based) applications in the file `/opt/ignite/data/examples/noncore.cfg`. The following is an excerpt from this file:

```
#############################################################
##
## Filename:    noncore.cfg
##
## @(#) noncore.cfg $Revision: 10.10 $
##
##
## Description:
##
```

The template file provides a description of itself and how you can use it to define non-core operating system software installations.

```
##     If the software source is not an "SD" depot, then the
##     make_config command cannot be used to generate a config file
##     to represent it and therefore the config file would have to
##     be manually created.  This example gives a starting template
##     to use for such a config file.
##
##     This config file would be used for a non-coreOS (application)
##     tar/cpio archive.
##
```

When you copy this file do not make the name `noncore.cfg`, rather rename it to a name that allows you to observe what application it is defining. The following excerpt describes this concept:

```
##     This config file should be copied to:
##         /var/opt/ignite/data/Rel_B.<release>/<filename>
##     then edit this new file to match your situation.
##
##     The values that you will most likely need to change in this
##     new file are:
##         nfs_source   - IP address and directory path to server
##                        directory containing the archives.
##         archive_path - path name of archive to be loaded from the
```

```
##                         nfs_source location.
##         sw_source    - tag name of the software source.
##         sw_sel       - tag name of the software select.
##         description  - description of software.
##         impacts      - amount of space needed, for a given selection,
##                         on each named file system.
##
##     Also, you may need to:
##         - add or delete any sw_sel's you do not need.
##         - run /opt/ignite/lbin/archive_impact on each
##           archive and replace the "impacts" statements with
##           those given by the archive_impact tool.
##
##
##     When done editing this config file, be sure to run:
##         /opt/ignite/bin/instl_adm -T -f <filename>
##     to check for syntax errors.  Then either use manage_index, or
##     manually edit the /var/opt/ignite/INDEX file to add a reference
##     to the copy of the file you edited.
##
##     Note:
##         This example shows the use of an NFS mounted directory
##         from the Ignite-UX server to the client in order to obtain
##         access to the archives. You could also use the ftp_source or
##         remsh_source keywords instead.
##
##         In order to use the nfs_source keyword as shown, you must first
##         export the directory given in the nfs_source keyword. This  is
##         accomplished by editing the Ignite-UX server file /etc/exports
##         and adding the export information.  When editing is complete,
##         be sure to run the "/usr/sbin/exportfs -av" command (as root)
##         to make the take effect.
##
##     See also:
##         - instl_adm(4)        for syntax documentation
##         - instl_adm(1M)       for checking syntax (-T option)
##         - archive_impact(1M)  for generating "impacts" statements
##         - ignite(5)           for general overview
##
################################################################
```

The `sw_source` stanzas define the location of the archives that you want to reference using `sw_sel` clauses. In the following excerpt, the source format is set to archive so Ignite-UX will search the archive definitions in the `sw_sel` clauses that use this `sw_source`.

```
    ############################################################
    ##  Software Sources
    ############################################################

    sw_source "Per-Discipline Packs" {
```

```
        description = "Software Apps for Individual Disciplines"
        source_format = archive
        source_type = "NET"

        # Change this to be your NFS server's IP and path:
        nfs_source = "14.12.99.113:/var/opt/ignite/archives"
    }
```

The software categories, `sw_category` are defined next. Categories are a good way of grouping software together though if you create too many categories it may become more difficult to find software in the **Software** tab of the Ignite-UX GUI rather than easier. The following excerpt is an example of a `sw_category` definition:

```
    ############################################################
    ##  Software Categories
    ############################################################

    sw_category "Disciplines" {
        description = "Software for Individual Disciplines"
    }
```

Next is a test of the hardware model string. The expression is set to `9000/*` meaning that if the string returned contains `9000` followed by zero or more slashes (`/`) then the first set of software defined in the following excerpt is for PA-RISC systems only. The software definitions that follow the `else` are for Itanium®-based systems.

```
##
##  Note that the HARDWARE_MODEL clause is only needed if the sw_sels
##  inside it can only be installed on a certain type of machine.
##
HARDWARE_MODEL ~ "9000/*" {

    ######################################################
    ##  Software Selections
    ######################################################
```

The `sw_sel` clauses are self-explanatory. The important parts are:

- The src, which defines where the software is located.
- The archive type, which defines that it is a gzipped `tar` archive though it could be compressed or in `cpio` format.
- The archive name or path of the archive.
- The impacts statements that enable Ignite-UX to determine how much additional file system space is required for the application.

```
    sw_sel "EE pack" {
        description = "Software Apps for Electrical Engineering PA clients"
        sw_source = "Per-Discipline Packs"
```

```
        archive_type = gzip tar
        archive_path = "ee_client_archive_PA.gz"
        sw_category = "Disciplines"
        impacts = "/var" 12568Kb
        impacts = "/usr" 23468Kb
        impacts = "/" 2Kb
    }
```

The following `sw_sel` defines a different application, but shares all of the same attributes.

```
    sw_sel "ME pack" {
        description = "Software Apps for Mechanical Engineering PA clients"
        sw_source = "Per-Discipline Packs"
        archive_type = gzip tar
        archive_path = "me_client_archive_PA.gz"
        sw_category = "Disciplines"
        impacts = "/var" 23678Kb
        impacts = "/usr" 12986Kb
        impacts = "/opt" 56892Kb
        impacts = "/" 64Kb
    }
```

The following `sw_sel` is for Itanium®-based systems is very similar to the software product that definition for PA-RISC systems in the previous excerpts. The only differences are the `archive_path` names and the impacts statements. Notice that the software products have the same name since they cannot both be defined at the same time. Any software that depends on this software product can use the one name as a dependency (for example, in conjunction with corequisite or exrequisite).

```
} else { # IPF software selections:

    ########################################################
    ##  Software Selections
    ########################################################
    sw_sel "EE pack" {
        description = "Software Apps for Electrical Engineering IPF clients"
        sw_source = "Per-Discipline Packs"
        archive_type = gzip tar
        archive_path = "ee_client_archive_IPF.gz"
        sw_category = "Disciplines"
        impacts = "/var" 18852Kb
        impacts = "/usr" 35202Kb
        impacts = "/" 3Kb
    }
```

The next example `sw_sel` defines a different application, but shares all of the same attributes.

```
    sw_sel "ME pack" {
        description = "Software Apps for Mechanical Engineering IPF clients"
        sw_source = "Per-Discipline Packs"
        archive_type = gzip tar
        archive_path = "me_client_archive_IPF.gz"
        sw_category = "Disciplines"
        impacts = "/var" 35517Kb
        impacts = "/usr" 19479Kb
        impacts = "/opt" 85338Kb
        impacts = "/" 96Kb
    }
}
```

SD and archive bitness comparison

You may have noticed an issue with the non-SD application example. Special effort was taken to ensure that a 64-bit version of the product was not installed on a 32-bit system. You could set SD to do this automatically. On PA-RISC systems, you would set the SD attribute `os_name`. For example using `swlist -l fileset -a name -a os_name` to look at some of the installed filesets results in the following:

```
# FCMassStorage                          HP-UX
  FCMassStorage.FCMS-ENG-A-MAN           HP-UX
  FCMassStorage.FCMS-INIT                HP-UX
  FCMassStorage.FCMS-JPN-E-MAN           HP-UX
  FCMassStorage.FCMS-JPN-S-MAN           HP-UX
  FCMassStorage.FCMS-KRN                 HP-UX:*64
  FCMassStorage.FCMS-RUN                 HP-UX
  FDDI-KRN-COM                           HP-UX
  FDDI-KRN-COM.FDDI467-KRN               HP-UX:*64
...
```

Notice that `os_name` is set to `HP-UX:*64` for some of the filesets in the previous example. In the file, `/opt/ignite/data/Rel_B.11.11/config` that is included into the configuration clause that you setup is the following configuration code:

```
sd_command_line += " -x os_release=" +${release}
(_hp_os_bitness == "64")
{
    (!can_run_64bit)
    {
       ERROR += "This system model: \"" +${model}+ "\" is not supported for
running 64bit HP-UX, you must select the 32bit selection"
    }
    sd_command_line += " -x os_name=HP-UX:64 "
}
```

This code allows SD to select common and 64-bit only filesets (and not include 32-bit only filesets) into the operating system. Because there is nothing similar for archives, you must do this manually.

Adding the non-SD application configuration file to the INDEX file

Now you can easily add the application configuration file to our `cfg` clause in the configuration file using the following command:

```
manage_index -a -f /var/opt/ignite/data/Rel_B.11.11/myapps_cfg \
    -c "Custom B.11.11 installation using SD"
```

The software is now ready to install.

Installing patches from a depot

Installing patches from a depot is explained in the following three topics.

Setting up the depot

In this section, you will setup a depot containing the patches that you made sure were installed earlier to enable the system to read the DVD containing the core operating system (OE) software.

At this point, you have a depot containing the patches that is observed using `swlist`:

```
swlist -l depot

# Initializing...
# Target "test" has the following depot(s):
...
  /var/opt/ignite/depots/Rel_B.11.11/patches_iso
# swlist -d @ /var/opt/ignite/depots/Rel_B.11.11/patches_iso
# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.11/patches_iso
#

#
# No Bundle(s) on test:/var/opt/ignite/depots/Rel_B.11.11/patches_iso
# Product(s):
#

  PHCO_25841    1.0            Add Rock Ridge extension to mount_cdfs(1M)
  PHKL_26269    1.0            Rock Ridge extension for ISO-9660
  PHKL_28025    1.0            Rock Ridge extension for ISO-9660
```

Unfortunately, Ignite-UX does not work with products and filesets so you must package the product patches into bundles.

Packaging the patches into a bundle

Previously the use of the `make_bundles` command to package products into bundles was discussed; see "Package an application in SD format". The following command quickly places the patches into a bundle:

```
make_bundles -b -i -n "ISO_patches" \
-t "Patches to enable Rock Ridge natively" -r 1.0 \
/var/opt/ignite/depots/Rel_B.11.11/patches_iso

Generating list of unbundled filesets...

=======  10/11/04 23:37:34 EST  BEGIN swpackage SESSION

       * Session started for user "root@test.rc.aus.hp.com".

       * Source:        test:/var/tmp/psf.24606
       * Target:
         test:/var/opt/ignite/depots/Rel_B.11.11/patches_iso
       * Software selections:
           *

       * Beginning Selection Phase.
       * Reading the Product Specification File (PSF)
         "/var/tmp/psf.24606".
       * Reading the bundle "ISO_patches" at line 11.

       * Selection Phase succeeded.

       * Beginning Analysis Phase.
       * Analysis Phase succeeded.

       * Beginning Package Phase.
       * Packaging the bundle
         "ISO_patches,r=1.0,a=HP-UX_B.11.11_32/64,v=HP".
       * Package Phase succeeded.

=======  10/11/04 23:37:35 EST  END swpackage SESSION
```

You can see that the patches have all been placed into a bundle with the following command:

```
swlist -d @ /var/opt/ignite/depots/Rel_B.11.11/patches_iso

# Initializing...
# Contacting target "test"...
#
# Target:  test:/var/opt/ignite/depots/Rel_B.11.11/patches_iso
#

#
# Bundle(s):
```

```
#
  ISO_patches   1.0            Patches to enable Rock Ridge natively
```

Generating a configuration

Next, you must create a configuration file to describe the SD depot. This is very easy and only requires the `make_config` command:

```
make_config -s /var/opt/ignite/depots/Rel_B.11.11/patches_iso \
    -c /var/opt/ignite/data/Rel_B.11.11/patches_iso_cfg
```

The generated configuration file looks like the following:

```
#############################################################
##  Software Sources
#############################################################

sw_source "/var/opt/ignite/depots/Rel_B.11.11/patches_iso" {
    source_type = "NET"
    sd_server = "10.2.72.150"
    sd_depot_dir = "/var/opt/ignite/depots/Rel_B.11.11/patches_iso"
    source_format = SD
}

#########################################################
##  Other Software
#########################################################

sw_sel "ISO_patches" {
    description = "Patches to enable Rock Ridge natively"
    sw_source = "/var/opt/ignite/depots/Rel_B.11.11/patches_iso"
    sw_category = "Uncategorized"
    sd_software_list = "ISO_patches,r=1.0,a=HP-UX_B.11.11_32/64,v=HP"
    (_hp_os_bitness == "32") {
        impacts = "/sbin" 224Kb
        impacts = "/usr" 184Kb
    }
    (_hp_os_bitness == "64") {
        impacts = "/sbin" 224Kb
        impacts = "/usr" 224Kb
    }
}
```

The last step is to add the configuration file to the `cfg` clause you are assembling in the `INDEX` file using the following command:

```
manage_index -a -f /var/opt/ignite/data/Rel_B.11.11/patches_iso_cfg \
```

```
    -c "Custom B.11.11 installation using SD"
```

Customizing configuration

In the current INDEX file, you have the following cfg clause:

```
cfg "Custom B.11.11 installation using SD" {
        description "This selection supplies the default system configuration
that HP supplies for the B.11.11 release."
        "/opt/ignite/data/Rel_B.11.11/config"
        "/opt/ignite/data/Rel_B.11.11/hw_patches_cfg"
        "/var/opt/ignite/data/Rel_B.11.11/core_cfg"
        "/var/opt/ignite/data/Rel_B.11.11/myapps_cfg"
        "/var/opt/ignite/data/Rel_B.11.11/apps_1203_cfg"
        "/var/opt/ignite/data/Rel_B.11.11/patches_iso_cfg"
        "/var/opt/ignite/config.local"
}
```

You have a rich feature set to build on top of, the advantage to this is that the config and hw_patches_cfg files provide a good working base on which you can build a generic system or a customized system.

To begin customizing the system you must create a new configuration file then add it to the cfg clause with the following commands:

```
touch /var/opt/ignite/data/Rel_B.11.11/custom_cfg
manage_index -a -f /var/opt/ignite/data/Rel_B.11.11/custom_cfg \
    -c "Custom B.11.11 installation using SD"
```

You have a file to which you can start adding configuration information to customize the existing configuration files. HP does not recommend that you customize configuration files generated by Ignite-UX commands. If the software in the depot changes, it is much easier to recreate them using Ignite-UX commands than it is to maintain them manually.

For example, rather than changing the configuration files that define a software product to set it to be installed you can add configuration information to this new file to install the software just for this cfg clause. This means the configuration file that describes the depot could be used in other cfg clauses as well to define the contents of a depot without having to take additional action. You could add the following configuration information:

```
init sw_sel "B3901BA"=TRUE
init sw_sel "B5725AA"=TRUE
sw_sel "ISO_patches"=TRUE
```

This would install the ANSI/C compiler, the complete version of Ignite-UX, and the patch bundle you created previously onto the system being installed. Users running the Ignite-UX GUI to interactively change the software to be installed could change whether the

ANSI/C compiler or Ignite-UX is installed. The user has no choice in the installation of the ISO patches you packaged earlier since the software selection is set to `TRUE` without using `init`, which means that the Ignite-UX GUI cannot change the selection.

Next, you could define a custom disk layout that is a choice between the existing disk layouts defined in `/opt/ignite/data/Rel_B.11.11/config` or completely replacing them so you do not have a choice of any of the default disk layouts. To implement this choice you would add the following:

```
_hp_disk_layout = {
    "MyApp custom disk layout"
}
```

None of the default disk layout choices defined in `/opt/ignite/data/Rel_B.11.11/config` would be available (although not shown here you must define the disk layout that would be used).

Alternatively, you could use the following example that adds the new disk layout to list of available choices and then sets it to be the default choice (anyone installing a system could still use the other layouts defined in the `/opt/ignite/data/Rel_B.11.11/config`).

```
_hp_disk_layout += {
    "MyApp custom disk layout"
}
init _hp_disk_layout="MyApp custom disk layout"
```

It is important to use a good foundation, building on top of the release default configuration files and the depot (software) configuration files enables you to easily create more complex configurations.

If you maintain the core operating system (OE) and the applications media in different depots, you may need to make customizations for the bundles you have created that define patches. If some products are installed from the applications depot instead of the core operating system (OE) depot and you have patches that depend on the filesets in the bundles that are taken from the applications depot, you may need to set the `load_order` on `sw_sel` clauses describing patches to be `>10`. This guarantees that the patches are loaded after all software from the applications depot is installed.

You can do this by directly modifying the configuration files created to describe the patches or by adding a `sw_sel` similar to the following to the customization file; not in the configuration file that actually defines the patches:

```
sw_sel "ISO_patches" {
    load_order=15
}
```

This takes advantage of something that described earlier. If you have a `sw_sel` mentioned twice, the information defined in the second clause overrides the information previously defined. In this case, no `load_order` was defined for the `sw_sel`, `ISO_patches` so the change adds the existing definition to change its load order since the default is `5`.

A `load_order` of `10` ensures that the patches are loaded after the core operating system (OE), which is loaded at `load_order 0`, and the applications that default to `load_order 5`.

Installing with SD wrap-up

The previous sections illustrated the following concepts:

- How simple it can be to use SD to install systems; installation from an Ignite-UX server over the network was employed in the examples presented.
- How to setup your Ignite-UX server for use with SD installations.
- How to manage your configuration explaining some of the issues that you face when managing your configuration.
- Why it is a good idea to separate configuration that belongs to depots or archives from the default configuration supplied with Ignite-UX and custom configuration that you may layer over the default.
- Where possible you should not modify the configuration files created by Ignite-UX commands. If you change the software in a depot, you will probably want to recreate the configuration file that describes the depot; if you have manual changes you must review the file and reapply the same changes to the new configuration file. This is time consuming so keeping the configuration separate means you only have to be aware of changes and review the customizations layered on top of the other configuration files periodically.

Installation configurations using golden images

It is important to look at how to create the `cfg` clause used for a golden image installation in the `INDEX` file.

Golden image configuration file explanation

Provided with Ignite-UX is a configuration file that you can use to define a golden image installation. This file is `/opt/ignite/data/examples/core11.cfg` and the following is an example of this file:

```
##############################################################################
##
## Filename:    core11.cfg  (template for B.11.* systems)
##
## @(#) core11.cfg $Revision: 10.10 $
##
## Description:
##     This is an sample configuration file that enables an OS archive
##     to be installed via Ignite-UX.
##
##     Before Ignite-UX can use an OS archive like that created by
##     make_sys_image(1M), the following must be done:
##
```

```
The instructions for how to use this config file are actually located in it as
comments (see below).

##          - Make an OS archive of the desired system using
##            /opt/ignite/data/scripts/make_sys_image
##            (see make_sys_image(1M) for details).
##
##          - Determine if you want to use ftp, NFS, or remsh to access
##            the OS archive (stored on the Ignite-UX server).
##
##          - Make a copy this sample configuration file.
##
##          - Edit your copy of this configuration file and modify:
##              - The IP addresses of the Ignite-UX server.
##              - The paths to OS archives.
##              - The sw_sel keyword descriptions.
##              - remove any extra sw_sel clauses.
##              - run /opt/ignite/lbin/archive_impact on each OS archive
##                and replace the "impacts" statements with those given
##                by the archive_impact tool.
##
##          - Edit the /var/opt/ignite/INDEX file.  If you already have a
##            stanza in the INDEX file that refers to an SD depot, then
##            you will probably want to make a new stanza ("cfg" clause)
##            that does not have references to the core SD cfg files, and
##            insert the path to your newly edited configuration file.
##            The end result should look something like this:
##
```

HP recommends that `/opt/ignite/data/Rel_B.xx.yy/config` is included first, similarly with SD installs there is information already defined that allows you to start with a very strong configuration that you can customize for your environment.

It is easiest to begin customizing by renaming `core11.cfg` to a new name and placing it in `/opt/ignite/data/Rel_B.xx.yy/config`. It is likely that you will have more than one golden image installation defined for any one HP-UX revision so customizing `core11.cfg` directly is not recommended.

```
##                cfg "HP-UX via an OS archive" {
##                    description "11.00 system using archives."
##                    "/opt/ignite/data/Rel_B.11.00/config"
##                    "/opt/ignite/data/Rel_B.11.00/core11.cfg"
##                    "/var/opt/ignite/config.local"
##                }
##
##          - If you have additional archives or applications in an SD
##            depot that you would like loaded along with this archive,
##            you may create a configuration file for them and add it to
##            the INDEX file entry as well.
##
##            See make_config(1M) for creating configuration files for an
##            SD depot.  See the example file "noncore.cfg" for setting
```

```
##           up application archives.
##
##        -  After making changes, always run:
##
##            # /opt/ignite/bin/instl_adm -T
##
##           to check the syntax.
##
#############################################################################

#############################################################################
##  Software Sources
#############################################################################
```

In the `sw_source` clause for the core operating system archive, the source format is `archive` so `sw_sel` definitions that use this `sw_source` must be archives as well. Since this archive contains the core operating system, it must be loaded at load order zero (`0`)[64].

```
sw_source "core archive" {
    description = "HP-UX Core Operating System Archives"
    load_order = 0
    source_format = archive
    source_type="NET"

    # When using this configuration file with a tape or CD-ROM where the
    # archive is on the same piece of media, then un-comment the next
    # line to prevent a prompt to change media during install.
    # change_media=FALSE
```

The following instructions, the two scripts given as the `post_load_script` and the `post_config_script` are required and you should not remove them.

However, there may be instances in which you may need to change the contents of the two scripts (for more information, see "Instances that may require modifying **`os_arch_post_l`**"). If you need to modify these files, you must never change the versions under `/opt` because when you upgrade Ignite-UX new versions of the files are installed in that location. In other words, your modifications are overwritten and are lost. Instead, you should copy them to `/var/opt/ignite/scripts` and then modify them. Note that if you have multiple golden images that require different changes in `post_load|config_scripts`, you should rename the scripts and then reference the new names for each golden image installations.

Periodically after upgrading Ignite-UX you should check to see what differences exist between the new scripts in `/opt/ignite/data/scripts` and any scripts you may have created using these examples. Ignite-UX may deliver enhancements or defect fixes that

---

[64] Only one archive can be loaded at load order zero (0), and it must be the core operating system archive. For technical reasons if the core operating system archive cannot fit within a certain size (cannot be written to a CD) then you must attempt to split out the software that can be loaded after load order one. For example, `/opt`.

you need in these scripts. If you want to take advantage of these updates, you must manually merge the changes.

```
    # The following 2 scripts should be reviewed to see if the defined
    # defaults are applicable.  If you need to modify them, copy them
    # to /var/opt/ignite/scripts first and change the paths here.
    post_load_script = "/opt/ignite/data/scripts/os_arch_post_l"
    post_config_script = "/opt/ignite/data/scripts/os_arch_post_c"

    #   If nfs_source is used, be sure to export the source.
    nfs_source = "10.2.72.150:/var/opt/ignite/My_Golden_Images"

    # ftp_source and remsh_source are alternate ways to copy the
    # archive.  For example:
    #   ftp_source = "anonymous@10.2.72.150:myusername"
    #   remsh_source = "myusername@10.2.72.150"
}

################################################################################
##  Software Categories
################################################################################
```

Next, categories that define the software that can be installed using the Ignite-UX GUI are defined.

```
sw_category "Languages" {
    description = "Languages"
}

sw_category "HPUXEnvironments" {
    description = "HP-UX Operating Environments"
}
```

Note that there are no Operating Environment (OE) definitions. This is not possible because software packaged in SD format is required.

```
################################################################################
##  Operating Environments
################################################################################

################################################################################
##  OS archives
################################################################################
```

Next is an example 32-bit operating system archive `sw_sel`. Note that the logic in this stanza dictates that if the system can run 32-bit HP-UX, it becomes the default golden image to be installed. If you want to change this test so that any system that could run 64-bit HP-UX, you would replace the logic that sets the `sw_sel` to `TRUE` or `FALSE` in the

`golden image - 32 bit OS` clause with `!can_run_64bit` and in the `golden image - 32 bit OS` clause replace the final test with `can_run_64bit`.

The use of `exrequisite` prevents users from choosing the 32-bit and the 64-bit core operating systems to load at the same time. Also, the use of `visible_if` prevents one or the other core operating system archive from appearing if the system can only run 32-bit or 64-bit HP-UX but not both. This is a good practice since you should not confuse users performing installations by offering software that cannot be installed onto the system as a choice.

If you are installing HP-UX B.11.23, you would remove the 32-bit `sw_sel` (since HP-UX B.11.23 for PA-RISC systems does not provide a 32-bit kernel). If you needed to support a mixture of PA-RISC and Itanium®-based systems, consider placing it with the definition of an archive for Itanium®-based systems.

```
init sw_sel "golden image - 32 bit OS" {
    description = "English HP-UX 11.00 CDE - 32 Bit OS"
    sw_source = "core archive"
    sw_category = "HPUXEnvironments"
    archive_type = gzip tar

    # For NFS, the path to the archive is relative to the mount point
    # specified in the "nfs_source" keyword within the sw_source stanza
    # above:
    archive_path = "hp_client1_B.11.00_32bitCDE.gz"

    # ftp and remsh sources can use a full path:
    # archive_path = "/pub/IUXarchives/hp_client1_B.11.00_32bitCDE.gz"

    # The data for the "impacts" statements are found by running:
    # /opt/ignite/lbin/archive_impact -t -g <OS_archive>
    impacts = "/"         27Kb
    impacts = "/.dt"      35Kb
    impacts = "/etc"    1864Kb
    impacts = "/export"    1Kb
    impacts = "/home"      1Kb
    impacts = "/opt"   74096Kb
    impacts = "/sbin"  13449Kb
    impacts = "/stand"     1Kb
    impacts = "/tmp"       1Kb
    impacts = "/users"    40Kb
    impacts = "/usr"  225951Kb
    impacts = "/var"    5705Kb
    exrequisite += "golden image - 64 bit OS"
    visible_if  = can_run_32bit
} = (can_run_32bit)
```

Following is the example for a 64-bit golden archive. All of the information previously presented regarding the 32-bit golden image are applicable to a 64-bit golden archive as you can see in the example.

```
init sw_sel "golden image - 64 bit OS" {
```

```
    description = "English HP-UX 11.00 CDE - 64 Bit OS"
    sw_source = "core archive"
    sw_category = "HPUXEnvironments"
    archive_type = gzip cpio
    archive_path = "hp_client1_B.11.00_64bitCDE.gz"
    impacts = "/"       421Kb
    impacts = "/sbin" 30086Kb
    impacts = "/opt"  78654Kb
    impacts = "/usr" 276420Kb
    impacts = "/var"  10059Kb
    exrequisite += "golden image - 32 bit OS"
    visible_if  = can_run_64bit
} = (!can_run_32bit)
```

You must set `_hp_os_bitness`; this variable is required by Ignite-UX to determine the "bitness" of the kernel being installed. Depending on which archive has been selected, this variable is set to the corresponding value.

Tip:
If you install HP-UX B.11.23, you must unconditionally set `_hp_os_bitness` to `64`.

```
#############################################################################
## Set the variable _hp_os_bitness depending on which of the archive
## sw_sels was selected.  This variable is used by configuration files
## generated via make_config to determine which version (32-bit or
## 64-bit) of a bundle will be loaded.
#############################################################################
(sw_sel "golden image - 32 bit OS") {
    _hp_os_bitness = "32"
}
(sw_sel "golden image - 64 bit OS") {
    _hp_os_bitness = "64"
}
```

Following is a definition for the `English` language. The `sw_source` associated with it is a dummy `sw_source`. In other words, the source format is `cmd` and does not have any scripts so it does nothing. Note that the `sw_sel` does set the locale that you can choose from in the Ignite-UX GUI.

```
#############################################################################
## Languages
## Specified so that the selector in the UI will not be empty.
## Cannot use the "core archive" sw_source because there are no
## archives associated with it.
##
## The "no select" sw_source is a no-op source since it is "cmd"
## format, and does not have any scripts specified.  Every sw_sel needs
## a sw_source.
```

```
#########################################################################
sw_source "no select" {
    source_format = cmd
}

init sw_sel "English" {
    description = "English Language Environment"
    sw_source = "no select"
    sw_category = "Languages"
    locale = { "SET_NULL_LOCALE:English", "C:English" }
} = TRUE
```

If you wish to be able to select, for example, Japanese locales during installation, you would have to customize the configuration file to describe the locales available in the golden image. For example, you could replace the locale line above with the following configuration:

```
    locale = { "ja_JP.SJIS:Japanese", "ja_JP.eucJP:Japanese",
"ja_JP.utf8:Japanese", "SET_NULL_LOCALE:English", "C:English" }
```

This would allow you to select any of the Japanese or English locales available on the system. Note that the "English" description can be misleading. The locale `SET_NULL_LOCALE` means the default system locale (POSIX/C) and C are explicitly the POSIX/C locale.

The locale line in the configuration file should not be wrapped; rather, it should be on one contiguous line. You should never add locales that do not exist in the golden image to the configuration.

The format of a entry in the locale list is:

```
"<locale>:<description>"
```

Where `<locale>` is the name of the locale shown in the output of the `"locale -a"` command on the system on which the golden image was created (for example `ja_JP.SJIS`). This is followed immediately by a colon and then by a description of the locale. The description cannot contain white space; if you need more than one word in the description, replace any white space by the underscore ("_") character. For example `"ja_JP.SJIS:Japanese_SJIS"` has the space between Japanese and SJIS replaced with an underscore. Remember that the locale list entry must be enclosed by double quotes, as seen above, when placed into a configuration file.

You should also be aware that even though a locale might not be listed, it only means that it cannot be selected via the Ignite-UX GUI. Any locale that was present when the golden image was created will still be available on any system installed using the golden image.

The next section defines default keyboards. You could add other keyboards depending on your environment or replace the defaults with those applicable to you.

```
#########################################################################
## Keyboards
## Newer machines use USB as the keyboard interface.  This example
```

```
## shows a way to handle both older (PS2) and newer (USB) keyboards.
## There are many other keyboard types available, see itemap(1m).
#############################################################################
has_ps2 {
    _hp_keyboard = {
        "Not_Applicable",
        "PS2_DIN_US_English",
        "PS2_DIN_US_English_Euro"
    }
    init _hp_keyboard = "PS2_DIN_US_English"
}

has_usb {
    _hp_keyboard = {
        "Not_Applicable",
        "USB_PS2_DIN_US_English",
        "USB_PS2_DIN_US_English_Euro"
    }
    init _hp_keyboard = "USB_PS2_DIN_US_English"
}
```

Instances that may require modifying **os_arch_post_1**

There are circumstances in which you may want to modify `os_arch_post_1`. For example, the following excerpt is from the `os_arch_post_1` script:

```
#
#  Networking files:
#    /etc/hosts
#    /etc/resolv.conf
#    /etc/rc.config.d/namesvrs
#       By default, these files will be constructed from the information
#       in the config file. The starting point for the hosts file is the
#       /usr/newconfig version, which just has a loopback entry. The other
files
#       are built from scratch. To get the archive versions of these files,
#       uncomment only the save_file lines here and comment out the rm line.
#       Using save_file will restore the file as-is from the archive. Using
#       merge_file will allow Ignite-UX to make changes to the file based
#       on what the config file or changes made in the UI.
#
#save_file /etc/hosts
#merge_file /etc/hosts
rm -f /etc/resolv.conf
#save_file /etc/resolv.conf
```

```
#merge_file /etc/resolv.conf
#save_file /etc/rc.config.d/namesvrs
#merge_file /etc/rc.config.d/namesvrs:
```

By default, during an installation [65] from an archive, the final versions of these files are taken from `/usr/newconfig`. If you have a complex configuration that needs to be preserved from the archive version of the file, you must uncomment the corresponding `merge_file` entry.

For example, if your `/etc/hosts` file contains extra entries that you need to preserve, you would uncomment the `merge_file /etc/hosts` line in your `os_arch_post_1` script. In the case of `/etc/hosts`, it may not be a good idea to use `save_file` as it means that any changes Ignite-UX made to `/etc/hosts` would be overwritten by the version saved from the archive.

In summary you can use:

- `save_file` to keep the version of the file from the Operating System archive preventing Ignite-UX from making changes to the file;
- `merge_file` to keep the version of the file from the Operation System archive but allow Ignite-UX to make customizations to it based upon the configuration for the installation session.

Creating a golden image

The creation of a golden image is not discussed in this white paper though for more information, see *`make_sys_image`*`(1M)` and the *Ignite-UX Administration Guide*.

It is important to realize that a golden image is an archive created by `make_sys_image` at clean level 2 (`make_sys_image -l 2`) so it contains no host-specific information where possible. You should not call a `make_net_recovery` or `make_sys_image` archive created at any other clean level for a golden image. In other words, to be a golden image, the archive must be created at clean level 2 using `make_sys_image`.

Final words about golden image installations

If you look at the example `cfg` clause from the `/var/opt/ignite/INDEX` file described in the `core11.cfg` file, you can see similarities between it and the SD-based `cfg` clause:

```
cfg "HP-UX via an OS archive" {
    description "11.00 system using archives."
    "/opt/ignite/data/Rel_B.11.00/config"
    "/opt/ignite/data/Rel_B.11.00/core11.cfg"
    "/var/opt/ignite/config.local"
}
```

[65] This does not apply to recovery situations. A `merge_file` is executed for the files in the `os_post_arch_1` script. You can find out what files maybe merged or saved automatically in a recovery situation by searching os_arch_post_l for "RECOVERY_MODE".

Both clauses include the release-specific configuration file `config` before any other configuration files are defined. Then the configuration file that defines the location of the core operating system (OE) is defined.

Because `config.local` must always be loaded last, you can see that after the entry for `core11.cfg` is the point at which you would begin adding additional configuration files to install other software like patches, additional applications, and customizations.

With virtually no changes, the contents, and methods explained in the "Installation configurations using Software Distributor depots" section can be used here. This is because the name of the core operating system `sw_sel` will change so some small changes are required if you have corequisites, exrequisities, or other dependencies on this name[66]).

Understanding what `is_net_info_temporary` does

Ignite-UX differentiates temporary and permanent network settings with the use of the final keyword. The final keyword is placed in front on networking attributes to indicate that they should apply to the final system. This behavior can change based upon the value of the `is_net_info_temporary` variable.

Information picked up from a DHCP/bootp server is placed into the configuration without the final keyword. Configuration files may also define network configuration without the final keyword. This configuration is treated in the same way as networking information picked up from a DHCP/bootp server.

When `is_net_info_temporary` is set to true, only networking information with the final keyword will be reflected in the network settings of the final system. When set to false (the default), networking information without the final keyword can override networking information specified with the final keyword.

To illustrate this, let's look at an example. The following partial configuration shows two DNS name servers defined for a system with the final keyword:

```
final dns_nameserver[0] = "10.1.1.10"
final dns_nameserver[1] = "10.1.1.30"
```

The entry from `/etc/bootptab` used to provide this system with network information when booting over the network looks like:

[66]The SD-based installation examples include a dependency between an application and which of the `sw_sels`, `HPUXBase32` or `HPUXBase64`,was selected. This would change to be based upon the `sw_sel` name of the core operating system (OE) golden images.

```
ignite-defaults:\
        ht=ethernet:\
        hn:\
        bf=/opt/ignite/boot/nbp.efi:\
        bs=48
ig0001:\
       tc=ignite-defaults:\
       ha=0073217D1429:\
       ip=10.1.1.119:\
       sm=255.255.255.0:\
       gw=10.1.1.1:\
       ds=10.1.1.10    *** DNS server
```

When booted using this `bootp` entry, Ignite-UX adds the following entry into the configuration automatically:

```
dns_nameserver[0] = "10.1.1.10"
```

With `is_net_info_temporary` set to false, the information Ignite-UX will write into `/etc/resolv.conf` for the DNS name server is:

```
nameserver 10.1.1.10
```

However with `is_net_info_temporary` set to true, the configuration marked with final is used instead:

```
nameserver 10.1.1.10
nameserver 10.1.1.30
```

This shows Ignite-UX using the networking configuration marked with the final keyword when `is_net_info_temporary` is set to true and using the networking configuration not marked with final when `is_net_info_temporary` set to false (in this case information picked up from a DHCP/bootp server).

If `is_net_info_temporary` is set to false, the following non-final networking information may be applied to the system unless the configuration already defines it with the final keyword:

```
- hostname
- DNS domain name
- NIS domain name
- NIS server
- NTP server
```

If `is_net_info_temporary` is set to false, the following networking information not marked final will be used instead of what is defined as final in the configuration:

```
- lan interface information (e.g. IP address, netmask, ...)
- DNS name servers
- DNS domain search list
```

If `is_net_info_temporary` is set to false, routing information not marked final will be applied to the system, but only if the configuration defines no final routing information.

This information is accurate as of Ignite-UX version C.6.6 (but it has not changed in a long time before that release of Ignite-UX).

If you encounter issues with any of the above information changing please investigate how you have set `is_net_info_temporary` (to true or false) and decide how it should be set to achieve the result you require.

How do I...

There are many times when questions arise regarding the use of Ignite-UX and the many utilities provided by the product. This section provides the answers to some of the more common questions that can occur.

How do I remove the warning message that occurs when compiling a kernel on PA-RISC systems?

When compiling a kernel in Ignite-UX on PA-RISC systems, it is likely that the following message from the linker, when it is linking the kernel, will appear:

```
/usr/ccs/bin/ld: (Warning) Linker features were used that may not be supported
in future releases.  The +vallcompatwarnings option can be used to display
more details, and the ld(1) man page contains additional information.  This
warning can be suppressed with the +vnocompatwarnings option.
```

You can prevent this warning by adding the following to your custom configuration:

```
# Stop the kernel linker warning from printing # when compiling a new kernel
is_hppa {
    env_vars += "LD_OPTS=+vnocompatwarnings"
}
```

How do I recognize if a disk exists or not from within a configuration file?

It is easy to see if a disk exists or not within an Ignite-UX configuration file. For example:

```
( disk[0/1/0/8/0.15.0].size == 0 ) {
        error += "Disk at 0/1/0/8/0.15.0 does not exist"
}
```

The previous example configuration excerpt adds an error message if the disk does not exist. This allows you to create a configuration that will alert you when disks that should be there are not. However, this same `error` keyword will prevent an actual install from occurring; so use this only if it is an actual fatal problem.

If you want to be alerted conditionally if the disk does exist, the example changes to the following:

```
( disk[0/1/0/8/0.15.0].size != 0 ) {
    ...
}
```

You could use this logic to conditionally define a disk layout only if the disks that it refers to exist. This would allow you to create unique disk layouts for various systems that have disks at different hardware addresses.

The size of a disk that does not exist on the system should always evaluate to zero. However, some disk arrays may present zero size disk LUNs. If there is a possibility of encountering this issue then you should consider the following example as an alternative (note that a zero length LUN is not be useful in a disk layout definition):

```
( (disk[0/1/0/8/0.15.0].model ~ ".") ) {
    ...
}
```

The logic inside the inner parenthesis tests whether the model string is set or not. The ~ does an extended regular expression match and the "." means match (one of) any character. Using this logic ensures that there is a disk LUN present at that address because it contains a model string so the configuration within the braces is evaluated only if the disk does exist.

You can change the sense of the test by adding a logical not operator (!):

```
( ! (disk[0/1/0/8/0.15.0].model ~ ".") ) {
    ...
}
```

Outside the inner parenthesis the ! (logical not) changes the sense of the test so it becomes TRUE if no model string is reported for that hardware path. This test could be used to perform actions if the disk did not exist.

For a more complete test, you could combine the previous examples as follows:

```
( (disk[0/1/0/8/0.15.0].model ~ ".") & (disk[0/1/0/8/0.15.0].size != 0) ) {
        init _my_var1=disk[0/1/0/8/0.15.0].model
        init _my_var2=disk[0/1/0/8/0.15.0].size
        note += "Disk at 0/1/0/8/0.15.0 has a model of " + ${_my_var1}
        note += "Disk at 0/1/0/8/0.15.0 has a size of " + ${_my_var2} }
```

This test ensures that the disk at the given hardware path has a model string and a non-zero size. If either test fails, the configuration it protects will not be evaluated. On a test system, this configuration produced the following output:

```
NOTE: Disk at 0/1/0/8/0.15.0 has a model of SEAGATE_ST39173WC
NOTE: Disk at 0/1/0/8/0.15.0 has a size of 8891556K
```

This result is because the disk did exist. The logic that you would use to see if a disk does *not* exist would be as follows:

```
( ! (disk[0/1/0/8/0.15.0].model ~ ".") & (disk[0/1/0/8/0.15.0].size ==
0) ) {
    ...
}
```

How do I create the CD equivalent of a tape created by **make_boot_tape**?

Although this question does not appear to be related to custom configuration, you can place custom configuration in the installation file systems contained on a custom installation CD that you create.

The following example instructions were developed using Ignite-UX version B.5.4. These instructions are applicable to Ignite-UX version C.6.0 and greater, however you will need to select an operating system revision using the `-r` option.

With Ignite-UX version B.5.4 this method creates a CD that can boot HP-UX 11.0 and B.11.11 systems and Itanium®-based systems running HP-UX B.11.23 (running OEs released before September 2004). With Ignite-UX C.6.0, you are queried for the HP-UX release so you can create a CD that supports Itanium®-based and PA-RISC systems running HP-UX B.11.23; you must create a different CD to support PA-RISC systems running HP-UX 11.0 and B.11.11.

You must install the correct Ignite-UX software to support the HP-UX revision for which you wish to create these CDs. In other words, if you install the HP-UX 11.0/B.11.11 bundles for Ignite-UX C.6.0 you will not be able to create a CD supporting B.11.23 since you have not installed the software supports that release of HP-UX.

Create a LIF. If you are using Ignite-UX versions before C.6.0, enter:

```
make_medialif -m -l ./my_lif
```

If you are using Ignite-UX versions C.6.0 and later, enter:

```
make_medialif –m –r B.XX.YY –l ./my_lif
```

where, B.XX.YY is the release of HP-UX for which you want to create the LIF. See Column 1 in Table 2 for a list of HP-UX releases you can use and what it will allow you to install or recover.

To verify the lif contents, use the following commands:

```
lifls -l ./my_lif
```

```
volume ISL10 data size 804676 directory size 3 04/11/15 13:57:02
filename   type   start   size     implement  created
============================================================
ISL        -12800 16      306      0          04/11/15 13:57:03
AUTO       -12289 328     1        0          04/11/15 13:57:03
HPUX       -12928 336     848      0          04/11/15 13:57:03
FWWKAR4    BIN    1184    1        0          04/11/15 13:57:03
FWWKAR5    BIN    1192    1        0          04/11/15 13:57:03
FWWKAR6    BIN    1200    1        0          04/11/15 13:57:04
FWWKAR7    BIN    1208    1        0          04/11/15 13:57:04
FWWKAR8    BIN    1216    1        0          04/11/15 13:57:04
INSTALL    -12290 1224    68184    0          04/11/15 13:57:09
INSTALLFS  -12290 69408   35840    0          04/11/15 13:57:12
VINSTALLFS -12290 69408   35840    0          04/11/15 13:57:12
WINSTALLFS -12290 69408   35840    0          04/11/15 13:57:12
IINSTALL   -12290 105248  197630   0          04/11/15 13:57:26
VINSTALL   -12290 302880  73806    0          04/11/15 13:57:32
WINSTALL   -12290 376688  83837    0          04/11/15 13:57:38
IINSTALLFS -12290 460528  116224   0          04/11/15 13:57:47
PAD        BIN    576752  256      0          04/11/15 13:57:48
```

If you have customizations to apply to the installation file systems, you should apply them now using `instl_adm`.

The `instl_adm` command , prior to Ignite-UX version C.6.0.*x*, will **not** update the Itanium®-based installation file system `IINSTALLFS` if any PA-RISC installation file systems are present (this issue is being addressed at this time; review the release notes for your version of Ignite-UX to determine if it has been resolved). If you wish to apply custom configuration to the Itanium®-based installation file system, you must perform the following commands:

```
lifcp ./mylif:IINSTALLFS ./IINSTALLFS
instl_adm -f output -F ./IINSTALLFS
lifrm ./mylif:IINSTALLFS
lifcp -r -T-12290 -K2 ./IINSTALLFS ./mylif:IINSTALLFS
```

This assumes that the configuration you wish to apply is located within the file called `output` above.

Remember that these steps are not required if the LIF does not contain any of the PA-RISC installation file systems (`INSTALLFS`, `WINSTALLFS`, or `VINSTALLFS`) or if you are using Ignite-UX C.6.0.*x* or greater. Note that with Ignite-UX C.6.0.*x* and greater, all of the installation file systems will be updated by `instl_adm`. If you wish to maintain different configurations in each filesystem, you must manually modify the Itanium-based installation filesystem, `IINSTALLFS` as previously described.

Tip:
If you only wish to support PA-RISC systems, you can write the LIF you just created directly onto a CD and use it to boot systems.

---

4. You must create a pseudo-root for our CD image; this is necessary to add support for Itanium®-based systems. Enter the following commands:

```
mkdir top
mv my_lif top
cp /opt/ignite/boot/EFI_CD_image top/EFI_CD_image
cd top
```

5. Create an ISO image that can be written to a CD. You may need to load additional software from your applications media to get the `mkisofs` command onto your system.

```
/opt/mkisofs/bin/mkisofs -no-emul-boot -b EFI_CD_image \
-eltorito-alt-boot -no-emul-boot -b my_lif -o ../cdfs.out $(pwd)
```

Messages similar to the following appear:

```
Size of boot image is 10240 sectors -> No emulation
Size of boot image is 288504 sectors -> No emulation
6.69% done, estimate finish Mon Nov 15 18:05:23 2004
13.39% done, estimate finish Mon Nov 15 18:05:31 2004
20.07% done, estimate finish Mon Nov 15 18:05:28 2004
26.77% done, estimate finish Mon Nov 15 18:05:27 2004
33.45% done, estimate finish Mon Nov 15 18:05:26 2004
40.15% done, estimate finish Mon Nov 15 18:05:26 2004
46.83% done, estimate finish Mon Nov 15 18:05:26 2004
53.54% done, estimate finish Mon Nov 15 18:05:25 2004
60.21% done, estimate finish Mon Nov 15 18:05:25 2004
```

```
66.92% done, estimate finish Mon Nov 15 18:05:25 2004
73.60% done, estimate finish Mon Nov 15 18:05:25 2004
80.30% done, estimate finish Mon Nov 15 18:05:25 2004
86.98% done, estimate finish Mon Nov 15 18:05:25 2004
93.68% done, estimate finish Mon Nov 15 18:05:25 2004
Total translation table size: 2048
Total rock ridge attributes bytes: 0
Total directory bytes: 0
Path table size(bytes): 10
Max brk space used 9000
74736 extents written (145 Mb)
```

6. Ensure that there are no `mkisofs` command errors. Older versions of `mkisofs` do not accept the `-no-emul-boot` option given twice, only accept the use of the `-b` option once, and would not accept the `-eltorito-alt-boot` option at all.

7. Place a LIF directory at the start of the ISO image with the `instl_combine` command so that the media will boot:

   ```
   /opt/ignite/lbin/instl_combine -C cdfs.out
   ```

   Itanium®-based systems do not require a LIF to be present to boot (PA-RISC systems do), but the kernel loader used to install Itanium®-based systems does read the installation kernel and file system from the LIF. Later in the installation process, other files will be read from the LIF.

   For more information regarding how to create custom installation media, refer to the *Ignite-UX Administration Guide*.

8. You can now write the ISO image onto a CD. The ISO image created in this example was written onto the CD using `Cdrecord` on an Itanium®-based system:

   ```
   cdrecord -data -v speed=4 dev=4,0,0 /var/tmp/cdfs.out
   ```

   It is important to realize that some Windows-based CD writing software cannot create CDs that will boot on Itanium®-based systems. The CDs produced by some CD writing software may boot PA-RISC systems, but fail to boot Itanium®-based systems.

This CD, like a tape created by `make_boot_tape`, allows you to boot a system locally so that it can contact an Ignite-UX server. In addition, this media is capable of performing a dual-media recovery. For more information regarding dual-media recovery, refer to the *Ignite-UX Administration Guide*.

How do I enable the X server and CDE during a golden image install?

The information provided here is intended only for golden image installs and not for SD-UX based installations. You should *not* use these commands during an SD-UX based installation. It is also assumed that you are using a version of Ignite-UX recent enough to support the graphics card in question (for newer graphics cards), and that the golden image has the required products and patches loaded to support the graphics card and input device.

The following configuration can be placed into a custom configuration file to enable CDE:

```
( graphics[].model ~ "." ) {
       post_configure_cmd+="cp -p /usr/dt/config/Xservers
/etc/dt/config/Xservers"
       post_configure_cmd+="/usr/dt/bin/dtconfig -e"
} else {
       post_configure_cmd+="/usr/dt/bin/dtconfig -d"
}
```

When a golden image is created on a system without a graphics card, CDE and the X server are disabled from starting. When the golden image is then loaded onto a system with a graphics card, CDE and the X server still won't start by default. The reason for this is that CDE disabled the X server from starting on the system without a graphics card where the golden image was created, and CDE then never checks again to see if any other system where the golden image is loaded has a graphics card.

CDE disables the X server by commenting out a line in `/etc/dt/config/Xservers`. Therefore, replace this file with `/usr/dt/config/Xservers` to make CDE check to see if the local X server can be started on its next attempt at starting up. During a golden image installation, this allows you to automatically enable the X server so CDE will start it after installation.

Please note the following limitation, though. The above configuration assumes that the commands above are all that are needed to enable the X server. The contents of the configuration files in `/etc/X11` may not be compatible with the newly installed system if they have been customized. You may need to replace files in `/etc/X11` to successfully get the X server started and working as desired. How to configure files in `/etc/X11` is beyond the scope of this document, however. For information on how to configure the X server files in `/etc/X11`, please review the SD-UX configuration scripts associated with the X server filesets located under `/var/adm/sw/products`.

Summary

A firm understanding of how to design, maintain, test and troubleshoot configuration files is essential in reducing business resources. With careful consideration and planning, you can create custom configuration files that improve mass deployments and system standardization.

For more information

The following relevant documents are available online at the HP Technical Documentation Web site at http://www.docs.hp.com/:

Ignite-UX Administration Guide

Successful System Cloning Using Ignite-UX

Managing HP-UX Software With SD-UX

Software Distributor Administration Guide for HP-UX 11i

HP-UX 11.0 Installation and Update Guide

Release Notes for HP-UX 11.0

HP-UX 11i v[1|1.6|2] Installation and Update Guide

HP-UX 11i v[1|1.6|2] Release Notes

Managing Systems and Workgroups: A Guide for HP-UX System Administrators

Additionally, there are a number of white papers pertinent to Ignite-UX that are located at the HP Technical Documentation Web site.

Product information regarding Ignite-UX for HP-UX is available at the HP Software Depot at

http://www.docs.hp.com/en/IUX/

MPN 5991-0691, 12/2005