HP-UX 10.X Startup and Configuration

1. Introduction

This paper explains the 10.X system startup and configuration models and outlines the differences between HP-UX Releases 9.X and 10.X. All of the changes apply to both S700 and S800 computer systems. These differences will primarily affect software developers and system administrators. Software developers will construct rc files in a different manner, specifying run-level execution order and enabling control through configuration variables. System administrators will control subsystem behavior, on a per-host basis, by modifying variables that control subsystem startup and shutdown.

1.1 Design Considerations for the New Startup and Configuration Models

The 10.X Startup and Configuration models are similar to the designs used by other major UNIX vendors. Important new features of the design include:

- The model separates the execution scripts from the configuration information required for execution. System administrators may easily modify the behavior of the startup/shutdown sequence by changing configuration variables.
- As a result of the separation of OS code from user-entered configuration values, the design solves the problem of correctly updating modified system configuration files.
- Individual subsystems may now be started or stopped on a per-runstate basis. This enables fine levels of control and subsystem separation.
- All subsystems are now treated uniformly at startup. There are well-defined interfaces for input parameters and exit values for startup scripts. This makes the design easy to extend and modify.

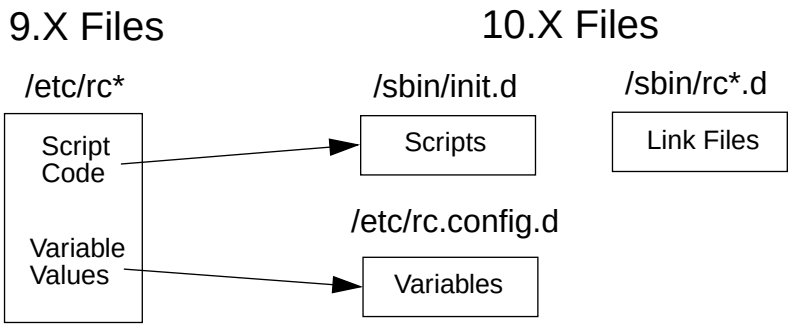
1.2 Startup/Shutdown Overview

The old **/etc/rc**-based startup and configuration design have been replaced in the HP-UX 10.X release. The new startup/shutdown design consists of three parts:

1. Execution Scripts used to startup and shutdown individual subsystems are contained in **/sbin/init.d**.
2. Configuration files for each execution script are contained in **/etc/rc.config.d** and reside in a filename that is generally equal to the script for which they configure.
3. Link Files in **/sbin/rc*.d** control the sequencing order of the execution scripts. The ordering of these link files should not be modified by the user.

FIGURE 1., “9.X - 10.X RC Configuration Mapping”, shows the relationship of 9.X files to 10.X files. Prior to 10.X, both script code and configuration information, such as IP addresses and hostnames, were contained within the files. The new model partitions the configuration data from the scripts. Administrators use the configuration variables in **/etc/rc.config.d** to change the behavior of scripts in **/sbin/init.d.** Script sequencing is also a new feature for 10.X and is controlled through link files in **/sbin/rc*.d** directories.

FIGURE 1. 9.X - 10.X RC Configuration Mapping



2. Startup and Configuration Models

This section describes the new startup and shutdown model components: execution scripts, configuration variable scripts and link files.

2.1 Execution Scripts

The **/sbin/init.d** directory contains all scripts used to startup and shutdown various subsystems. No script may invoke any of the other scripts in this directory. Scripts obtain configuration data from variables in **/etc/rc.config.d** (more on this later), which must be sourced by the execution script. All files in the **/etc/rc.config.d** directory may be read by sourcing the single file **/etc/rc.config**.

The startup and shutdown scripts shipped by HP in **/sbin/init.d** are not to be edited. Any changes made to these scripts will be overwritten when a new software release is installed. Modifying the behavior of a sub-system script is accomplished by using configuration variables, discussed in 2.3.4, “Configuration Files”.

In general, each script under **/sbin/init.d** should perform both the startup and shutdown functions. In order to control the functionality within the script, each must also support standard arguments and exit codes. Scripts must be written for the POSIX shell. A template script may be found in **/sbin/init.d/template.**

2.1.1 Arguments to scripts

The startup/shutdown scripts must recognize the following four arguments:

- **start_msg.** The “start_msg” argument is passed to scripts so the script can report back a short message indicating what the “start” action will do. For instance, when the lp spooler script is invoked with a “start_msg” argument, it echoes “Starting the LP subsystem”. This string is used by the startup checklist. Note that when given just the “start_msg” argument, scripts will only print a message and NOT perform any other actions.
- **start**. Upon receiving the “start” argument, the script should start the subsystem. All output should be echoed to *stdout*.
- **stop_msg**. The “stop_msg” argument is passed to scripts so that the script can report back a short message indicating what the “stop” action will do. For instance, when the lp spooler script is invoked with a “stop_msg” argument, it echoes “Stopping the LP subsystem”. This string is used by the shutdown checklist. Note that when given just the “stop_msg” argument, scripts will only print a message and NOT perform any other actions.
- **stop**. Upon receiving the “stop” argument, the script should shutdown the subsystem. All output should be echoed to *stdout*.

When passed the **start** and **stop** arguments, an execution script should not echo any messages indicating the entry or exit from the script. **start_msg** and **stop_msg** arguments will be passed to the execution scripts by **/sbin/rc** to record these messages both on the console and in log files, indicating the execution of the script.

2.1.2 Naming Conventions

The startup and shutdown scripts are named after the subsystem they control. For example, the **/sbin/init.d/cron** script controls the cron daemon.

2.1.3 Scripts and Console Output

To ensure proper reporting of startup events, startup scripts will be required to comply with a few guidelines for script output and exit values.

The messages echoed by the execution script when **start_msg** and **stop_msg** arguments are passed should contain a single line message with no more than 30 characters.

Status messages, such as “Starting Subsystem daemon”, must be directed to *stdout*. All error messages must be directed to *stderr*. Both *stdout* and *stderr* are redirected to the log file **/etc/rc.log**, unless the startup checklist mode is set to the raw mode. In this case, both *stdout* and *stderr* output go to the console.

Startup scripts, and the daemons or binaries they execute, must not send messages directly to the console during system boot or shutdown. This restriction exists because console output during the boot or shutdown sequence will overwrite the graphical checklist, leaving the checklist unreadable. These messages should be directed to *stdout* or *stderr*.

2.1.4 Exit values

Exit values for startup scripts are as follows:

0 -- script exited without error. This causes the status “OK” to appear in the checklist.

1 -- script encountered errors. This causes the status “FAIL” to appear in the checklist.

2 -- script was skipped due to overriding control variables from **/etc/rc.config.d** files or for other reasons, and did not actually do anything. This causes the status “N/A” to appear in the checklist.

3 -- script executed normally and requires an immediate system reboot for the changes to take effect. This is reserved for key system components.

These are the only exit values acceptable. Returning an arbitrary non-zero exit value from a command to indicate failure may cause the script to appear to have been skipped in the checklist, or may cause the system to reboot.

2.2 Configuration Variable Scripts

Instead of spreading configuration data throughout the various rc files in the system, configuration data is structured as a directory of files that allows developers to create and manage their own configuration files, without the complications of shared file ownership. The directory that holds the configuration variable scripts is **/etc/rc.config.d**. The configuration variable scripts will be sourced by the startup and shutdown execution scripts in **/sbin/init.d** during system startup and shutdown. This configuration information is used by the execution scripts to enable/disable and configure subsystems (such as IP addresses).

Examples of these variables include:

- HOSTNAME: Internet name of your system.
- IP_ADDRESS[0]: Internet address of your system, in dot format.

These configuration variables may be defined as either simple shell variables of type string, or may be defined as singly-indexed array variables of basetype string. The use of indexed variables adds a powerful, yet easy to understand and use, capability to the system. Consider a system that contains two (or more) network interfaces and may serve as a routing agent. The various network interfaces can be simply defined and manipulated as follows:

```
INTERFACE_NAME[0]=lan0
```

IP_ADDRESS[0]=15.13.186.87

SUBNET_MASK[0]=255.255.248.0

LANCONFIG_ARGS[0]=”ether ieee”

INTERFACE_NAME[1]=lan3

IP_ADDRESS[1]=15.27.233.2

SUBNET_MASK[1]=255.255.248.0

LANCONFIG_ARGS[1]=”ether ieee”

Each execution script may reference its variables in one of two ways. If the execution script references only the variables delivered with its product or fileset, it may explicitly source the file in **/etc/rc.config.d/<subsystem>.** If the script requires variables that are delivered by other products or filesets, it may just source **/etc/rc.config,** which is a script that sources all the files below **/etc/rc.config.d**. The master configuration file **/etc/rc.config** skips filenames named **“core”**, or containing the characters **[.,~#]** when it sources the contents of **/etc/rc.config.d/**, to avoid sourcing a binary core dump file or a backup file that may have been left in that directory by a system administrator.

There must be no requirements on the order of the files sourced. This means configuration files must not refer to variables defined in other configuration files, since there is no guarantee that the variable being referenced is currently defined. Also, there is currently no registry of variable names maintained, so developers should take care to avoid variable namespace collisions.

Configuration variable scripts are written for the POSIX shell *(***/usr/bin/sh** or **/sbin/sh**), and not the Bourne *sh, ksh,* or *csh*. In some cases, these files must also be read, and possibly modified by other scripts or the SAM program. For this reason, each variable definition must appear on a separate line, in the syntax:

variable=*value*

No trailing comments may appear on a variable definition line. Comment statements must be on separate lines, with the “#” comment character in column 1. An example of the required syntax for configuration files is given below.

```
# Cron configuration. See cron(1m)
#
# CRON: Set to 1 to start cron daemon
#
CRON=1
```

The name of a configuration script in **/etc/rc.config.d** corresponds to the names of the associated startup and shutdown scripts found in **/sbin/init.d**. In many cases the same 10-character base name may be used for the execution script, the sequencing links, and the configuration file. Here is an example using the cron subsystem.

/sbin/init.d/cron	execution script
/etc/rc.config.d/cron	configuration file
/sbin/rc2.d/S730cron	start sequence symbolic link
/sbin/rc1.d/K270cron	kill sequence symbolic link
/usr/sbin/cron	cron daemon (binary file)

The file **/etc/TIMEZONE** contains the definition of the TZ environment variable. Its location is defined by standards documents. It is sourced by **/etc/rc**.**config** along with the other **/etc/rc.config.d/*** files. The file **/etc/rc.config.d/LANG** contains the National Language Support language specification.

A more detailed listing of **/etc/rc.config.d/** configuration files is given in APPENDIX 2., “Configuration Files HP-UX Release 10.X”.

2.3 Sequencing Scripts with Link Files

The third aspect of the startup/shutdown design is controlling the order of script execution with link files. An important feature enables developers to control individual subsystems among run-levels. This is accomplished by providing symbolic link files in run level directories that point to the scripts in **/sbin/init.d**.

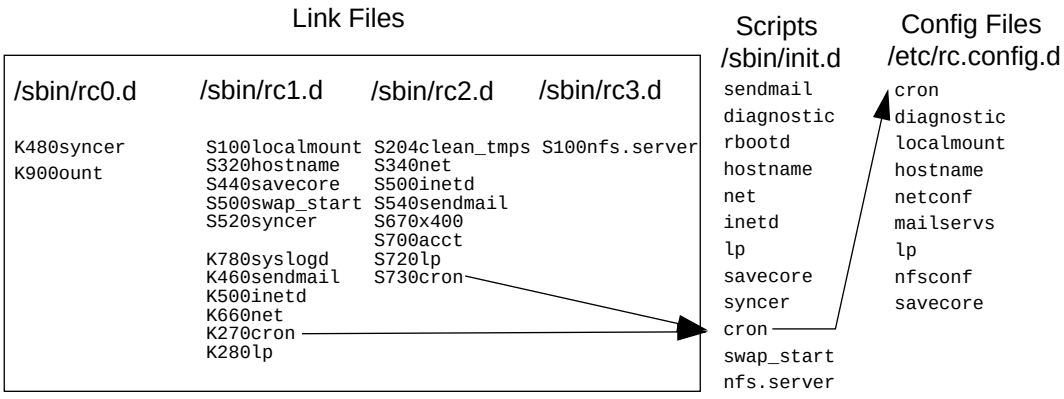
2.3.1 Run-level Directories: /sbin/rc#.d

The **/sbin/rc#.d** (where # is a run-level [0..6]) directories are startup and shutdown sequencer directories. They contain only symbolic links to startup/shutdown scripts in **/sbin/init.d** that are executed by **/sbin/rc** on transition to a specific run level. For example, the **/sbin/rc3.d** directory contains symlinks to scripts that are executed when entering run level 3. (There is more information on **/sbin/rc** in Section 2.4, “Run Levels and /sbin/rc”).

These directories contain two types of link files: *start* links and *kill* links. Start links have names beginning with the capital letter “S” and are invoked with the “start” argument at system boot time or on transition to a higher run level. Kill links have names beginning with the capital letter “K” and are invoked with the “stop” argument at system shutdown time, or when moving to a lower run level.

Further, all link files in a sequencer directory are numbered to ensure a particular execution sequence. Each script has, as part of its name, a three digit sequence number. This, in combination with the start and kill notation, provides all the information necessary to properly startup and shutdown a system**.**

FIGURE 2. Startup/Shutdown Component Relationships.



Note: The sequence numbers above are only for example and may not accurately represent your system.

Figure 2 shows the run-level directories and the relationship of the link files to the scripts. Because each script in **/sbin/init.d** performs both the startup and shutdown functions, each will have two links pointing towards the script from **/sbin/rc*.d**: one for the start action and one for the stop action.

2.3.2 Naming Conventions

The naming conventions for the link files are as follows:

/sbin/rc2.d/S730cron
1 2 3 4

The various components have the following meanings:

1. Run Level Number: The sequencer directory is numbered to reflect the run-level for which its contents will be executed. In this case, Start scripts in this directory will be executed upon entering run-level 2 from run-level 1, and Kill scripts will be executed on entering runlevel 2 from runlevel 3.
2. Sequencing Type: The first character of a sequencer link name determines whether the script is executed as a start script (if the character is “S”), or as a kill script (if the character is “K”).
3. Sequence Number: A three digit number is used for sequencing scripts within the sequencer directory. Scripts are executed by type (start or kill) in lexicographical order.
4. Script Name: Following the sequence number is the name of the startup script. This name must be the same name as the script to which this sequencer entry is linked. In this example, the link points to **/sbin/init.d/cron.**

Scripts are executed in lexicographical order. The entire file name of the link is used for ordering purposes. When adding new sequencer entries, sequencer numbers are chosen to allow for gaps so that future entries may be inserted without requiring renumbering of existing entries. (There is more information on sequence numbers in Section 2.5, “Link File Sequence Number Assignment”).

Subsystems are killed in the opposite order they were started. This implies that kill scripts will generally not have the same numbers as their start script counterparts. For example, if two subsystems must be started in a given order due to dependencies (e.g., **S111sys1** followed by **S222uses_sys1**), the counterparts to these scripts must be numbered so that the subsystems are stopped in the opposite order in which they were started (e.g.**, K555uses_sys1** followed by **K777sys1**).

Also, kill scripts for start scripts in directory **/sbin/rc**N**.d** reside in **/sbin/rc**(N-1)**.d**. For example **/sbin/rc3.d/S123system2** and **/sbin/rc2.d/K654system2** might be start/kill counterparts.

Core components of HP-UX will continue to support short filenames (i.e. 14 characters). Because of the filename place holders for ‘K’, ‘S’ and the 3 digit sequence numbers, subsystem script names in **/etc/init.d** should be restricted to 10 characters in length.

2.4 Run Levels and /sbin/rc

In previous HP-UX releases, **/etc/rc** was run only once. Now **/sbin/rc** may be invoked several times during the execution of a system, once on each occasion the init process changes run levels. **/sbin/rc** sequences through the execution scripts as it transitions through the run levels. However, only the subsystems configured for execution, through configuration variables in **/etc/rc.config.d**, are started or stopped when transitioning the run levels.

Upon transition from a lower to a higher run level, the start scripts for the new run level and all intermediate levels between the old and new level are executed. Upon transition from a higher to a lower run level, the kill scripts for the new run level and all intermediate levels between the old and new level are executed.

When a system is booted to a particular run level, it will execute startup scripts for all run levels up to and including the specified level (except run level 0). For example, if booting to run level 4, **/sbin/rc** looks at the old run level (S) and the new run level (4) and executes all start scripts in states 1, 2, 3, and 4. Each level is sorted and executed separately to ensure that the lower level subsystems are started before the higher level subsystems.

Consequently, when shutting down a system, the reverse takes place. The kill scripts are executed in lexicographical order starting at the highest run level and working down, as to stop the subsystems in the reverse order they were started. As mentioned earlier, the numbering is reversed from the startup order.

States 0 and S are special cases. When entering state 0, **/sbin/rc** will run start scripts in **/sbin/rc0.d**. Start scripts in state 0 are quick system administration scripts that prepare the system for a shutdown. When entering state 0 from a higher run level, the system is halted. When entering state S on startup, the init process does not read /etc/inittab and does not invoke /sbin/rc. Instead, init execs a shell at the system console and the system is in single user state. The following table summarizes the run level definitions.

Table 1: Run Level Definitions

Run level	State	Sequencer Dir	/sbin/rc interaction
0	Halted	/sbin/rc0.d	All start and kill scripts executed
S	Single User		
1	Minimal System Configuration	/sbin/rc1.d	When entering from lower state, all start scripts are executed. When entering from higher state, all kill scripts are executed
2	Multi-User	/sbin/rc2.d	
3	Exported File Systems	/sbin/rc3.d	
4	HP-VUE or CDE	/sbin/rc4.d	
5,6	Not currently used.	/sbin/rc5.d /sbin/rc6.d	

2.5 Link File Sequence Number Rationale and Assignment

The HP-UX 10.X operating system and its associated products have been structured using a paradigm that groups various related functions into the same run state. This is accomplished by reserving blocks of sequence numbers for these functions. The paradigm is described in the sections below. Some entries provide examples of subsystems that are started at a particular level. A more detailed listing of sequence numbers may be found in APPENDIX 1., “Start/Kill sequence numbers HP-UX 10.X”. Please consult an HP-UX 10.X system for a complete and accurate listing.

2.5.1 Run Level 1 Paradigm

Run level 1 provides core services such as mounting filesystems and configuring key system parameters.

0XX	reserved for temporary links
1XX	mount local filesystems
2XX	essential process initialization/kill
3XX	set essential system parameters (hostname, lan address)
4XX	set other system parameters (date, privilege groups)
5XX	start essential daemons (swapper and syncer daemons)
6XX-8XX:	not currently used
9XX	reserved for future expansion

2.5.2 Run Level 2 Paradigm

Run level 2 is the general multi-user run state where most services are started.

0XX	reserved for temporary links
1XX	software installation/configuration (SD)
2XX	essential local daemons and services, started before network startup (clean log/tmp files, syslogd)
3XX	network startup
30X	network tracing/logging must be first

	31X-33X	network link-level services (FDDI,ATM,Fiber,token ring)
	34X	traditional TCP/IP initialization (ifconfig,route,gateway,netmask,etc.)
	35X-39X	other network startup (x25, loopback daemon, naming daemon)
4XX	NFS/NIS initialization	
5XX-6XX	services built on top of network services (DCE,DFS,NCS, rbootd, NetLS, mail)	
	(Also client/server services: X font server, Kanji server)	
	500	inetd super-server
7XX-8XX	other local daemons/services (lp, cron, diagnostics, auditing, accounting, etc.)	
9XX	reserved for future expansion	
	900	"Generic" number for run state 2

2.5.3 Run Level 3 Paradigm

Run level 3 is the networked multi-user state. This run level is used to export file systems. Currently HP only supports NFS exports. Other vendors (Solaris) also do RFA exports here.

0XX	reserved for temporary links
1XX	NFS exports (NFS server)
2XX-8XX	not currently used
9XX	reserved for future expansion

2.5.4 Run Level 4 Paradigm

Run level 4 is reserved for graphical interface managers. Currently, HP-VUE or CDE is started in this run level, but is invoked as an entry in **/etc/inittab**. No start/kill links are currently shipped in run level 4.

3. Startup Displays and Logging

3.1 Startup Display on System Console

The 10.X startup display has adopted the screen-oriented interface used in the HP-UX 9.0 Instant Ignition product. As individual subsystems are started a status line is written to the system console; this is the string returned when the execution script is invoked with **start_msg** or **stop_msg**. The execution script returns a value of 0, 1, or 2 (or 3 for the special case of Reboot) which is displayed as **OK**, **FAIL**, or **N/A**. For terminals supporting HP terminal escape sequences, the table is displayed in screen-addressing mode; for other terminals the display is written in line (scroll) mode. A **busy/wait** status is flashed if a subsystem takes longer than 5 seconds to start up. An example of a startup display is as follows:

```
HP-UX Start-up in progress                                  Status
--------------------------                                  ------

Mount file systems ..................................... [ OK ]
Setting hostname ....................................... [ OK ]
Set privilege group .................................... [ OK ]
Display date ........................................... [ OK ]
Save system core image if needed ....................... [N/A ]
Enable auxiliary swap space ............................ [FAIL] *
Start syncer daemon .................................... [ OK ]

* - An error has occurred !
* - Refer to the file /etc/rc.log for more information.
```

3.2 Startup Logging

All startup messages are logged to the file **/etc/rc.log**. All execution script output should be directed to *stdout* or *stderr,* and not to */dev/console*. The log file **/etc/rc.log** is re-written on every system boot, so that it contains the history of all runstate transitions for the current bootup. The immediately previous log is preserved in the file **/etc/rc.log.old.**

4. Extending the Startup Model

4.1 Adding Subsystems

System administrators and software product developers may have a need to add their subsystem(s) to this model. This may be easily accomplished by following a few steps.

A good place to start is with the execution script. This may be derived from any of the system execution scripts found in **/sbin/init.d.** There is also a blank template in **/sbin/init.d/template**. The guidelines for execution scripts must be strictly followed such that the interface with **/sbin/rc** is correctly maintained. Any failure to do so will likely cause the execution script to fail.

A configuration file may also need to be added to **/etc/rc.config.d**. This file should contain a variable assignment that enables the user or system administrator to enable or disable the particular subsystem. The file should also contain any other variable assignments required by the subsystem during startup. The configuration file should not contain any executable script code other than variable assignments.

Selecting a run state and a sequence number should probably be done last. Many times it is difficult to determine exactly where the subsystem should fit until it has been coded and tested.

4.1.1 Selecting Sequence Numbers

All of the sequence numbers for HP products have been carefully assigned to ensure correct ordering, eliminate overlap and allow room for growth. Each of these products has registered with a central organization and has been given specific sequence numbers based on a number of factors about the product. When choosing sequence numbers for products or applications, system administrators and software product developers should adhere to the guidelines below.

Commercially available products may require specific sequence number ordering. Software developers should not arbitrarily choose an unused/unassigned number that happens to be absent from the system or table they are reviewing. Absent numbers that appear unused may have already been assigned by Hewlett-Packard to a product not yet released, or is not installed on a particular system. Developers are encouraged to contact HP during their development cycle to determine correct assignments.

If specific ordering is not a concern and the subsystem may be started at any point after system boot and initialization, run level 2, number 900 should be used for the start link and run level 1, number 100 for the kill link. These “Generic” numbers may be used by *any* product or application without registering with Hewlett-Packard. Even if there are multiple entries of this type, the links are still unique because the entire filename is significant in ordering the scripts.

4.2 Guidelines for Startup/Shutdown Scripts

Startup execution scripts are installed in the **/sbin/init.d** directory and sequencer links are installed directly into the **/sbin/rcN.d/** directories directly from the install media. The startup execution scripts and the sequencing links are not customer editable. No provisions are made to keep newconfig versions of the scripts; any changes made by a customer to a system execution script are overwritten and lost when an update occurs.

Administrators and users should remain aware of the following:

- Execution scripts must not be modified. If a script in **/sbin/init.d** is modified, the modification will be lost at the next update, when the script is overwritten with a new version.
- It is not necessary to remove scripts from the sequencer. If a script is removed from the sequencer, it will be replaced at the next update. Control variables in files within **/etc/rc.config.d** should be used to control whether a startup script is executed.
- Sequencer links must not be renamed. If a symbolic link in the sequencer directory is renamed (or renumbered), at the next update, a symbolic link with the original name will be installed. This will result in two copies of the same startup script appearing in the sequencer.

All HP-supplied startup scripts are meant to be present in sequencer directories as installed. They should not be removed or renamed (to change the relative sequencing).

4.3 Developing Startup/Shutdown Scripts

Developers should take care to carefully design and test the start and kill scripts for their products. A startup script that may work on one individual test system may cause intermittent or misleading failures on other systems that are configured differently unless the script has been robustly developed and tested. Here are several considerations a startup script developer should review when designing a script.

1. The script should be written for the POSIX shell (**/sbin/sh**), and not the Bourne shell, K-shell, or C-shell. The POSIX shell is the only shell that is guaranteed to be present on the root filesystem at system bootup.
2. Scripts should be carefully tested by the developer before they are installed on a system. Scripts may be tested by executing the script and explicitly reviewing the script input parameters and exit values. This may be easily done by executing the command:

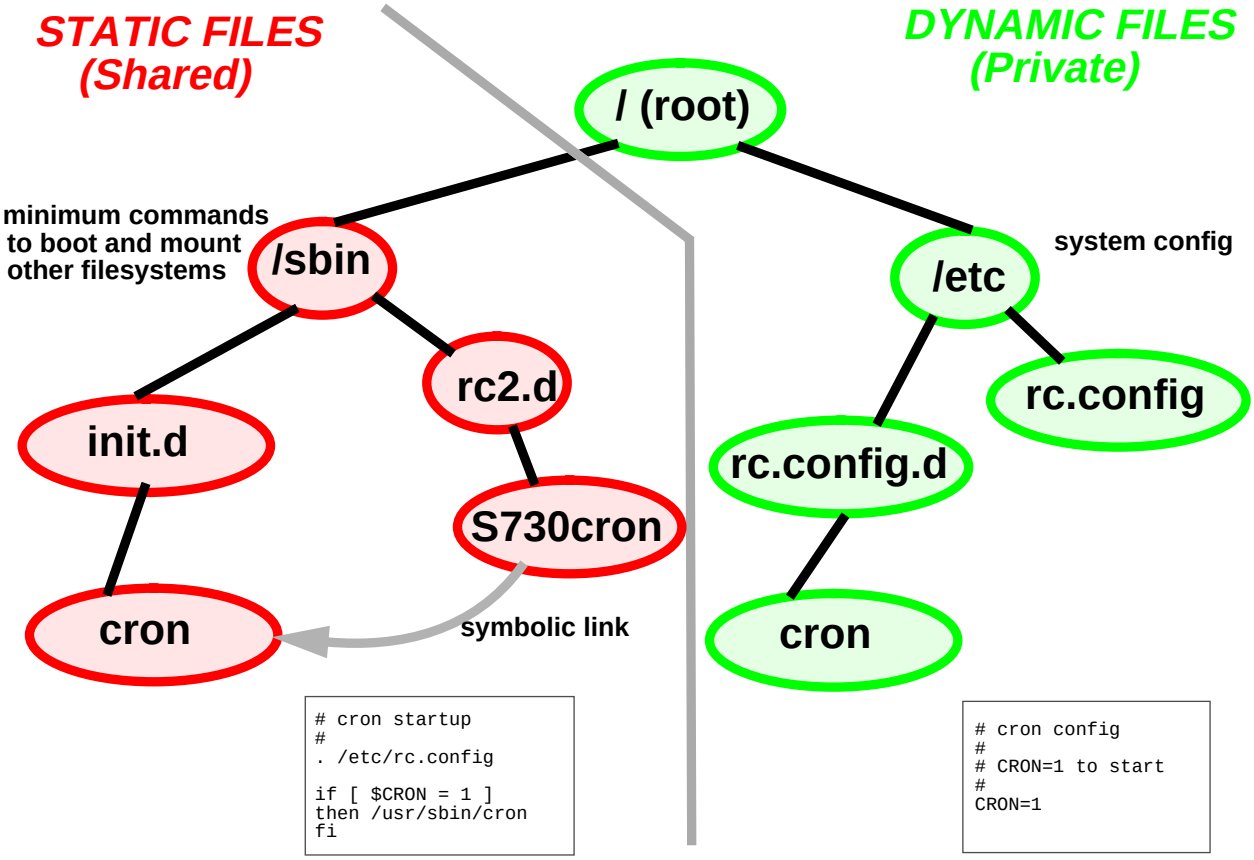
   ```
   % /sbin/sh -xv <your_startup_script> <input_parameters>
   ```

   where **<your_startup_script>** is the name of your script and **<input_parameters>** may have the possible values: **(start_msg, start, stop_msg, stop)**. For each of these invocations of the startup script under test, the script exit values should be verified to be one of the allowed exit values: **(0,1,2,3)**. Please refer to earlier sections of this paper for definitions of these input parameters and exit values.
3. Sequence numbers should be chosen carefully to not conflict with sequence numbers that may already be assigned. Even if a sequence number is not installed on your system in the directory **/sbin/rc[0-4].d/**, it may still be assigned to another product. Please refer to documentation in the directory /usr/share/doc for more complete listings of assigned startup sequence numbers.
4. Startup scripts must be written to use commands and libraries that are guaranteed to be present on the root filesystem of the system at bootup time. Startup scripts must not invoke any commands or libraries that are located under the **/usr**, **/opt**, or **/var** directories. In a multiple disk configuration, these directories are often located under filesystem mount points that are not available until late in the bootup cycle.
5. Startup scripts should be written to be portable across multiple product lines (S700 and S800), and as independent as possible of system configuration. A developer should test each startup script across a broad variety of systems and configurations.
6. Startup scripts should be written to work correctly when HP-UX is booted into any of the possible runlevels: **(S,1,2,3,4)**. A developer should test that each script continues to work correctly when HP-UX is brought up in each of these runlevels, and is transitioned between various combinations of these runlevels.

4.4 An Illustrative Example

Figure 3 below depicts a simple example for the startup of cron. The relationships of the files between the static and dynamic file systems is also shown.When entering run state 2 from a lower level, the ‘S’ scripts are executed. In the example, S730cron is a link to the cron script under /sbin/init.d. Cron will start because the configuration variable in **/etc/rc.config.d/cron** is set to 1. A value of 0 would not start cron.

FIGURE 3. Implementing cron in the New Model



4.5 File Name Conventions, Permissions, and Fileset Packaging

File name conventions have been presented in previous sections. Often, a developer may use the same 10-character file basename for the various startup file names and for associated daemons. Execution scripts are typically delivered (owner, group, permissions) = (bin, bin, 0555 read/execute), configuration files with permissions = 0444 read. All startup and configuration files should be delivered in the same fileset as the rest of the product, so that startup files may be installed and de-installed at the same time as the product is installed. This approach helps prevent invalid and potentially incorrect startup files from remaining on the system after a product is removed or updated. An example of file naming and fileset packaging is as follows.

Table 2: Startup File Naming and Packaging

Filename	Purpose	Product	Fileset
/sbin/init.d/lp	Execution script	PrinterMgmt	LP-SPOOL
/sbin/rc1.d/K280lp -> /sbin/init.d/lp	Start sequence link	PrinterMgmt	LP-SPOOL
/sbin/rc2.d/S720lp -> /sbin/init.d/lp	Kill sequence link	PrinterMgmt	LP-SPOOL
/etc/rc.config.d/lp	Configuration file	PrinterMgmt	LP-SPOOL
/usr/sbin/lpsched	Daemon (binary)	PrinterMgmt	LP-SPOOL
/usr/bin/lp	Command (binary)	PrinterMgmt	LP-SPOOL