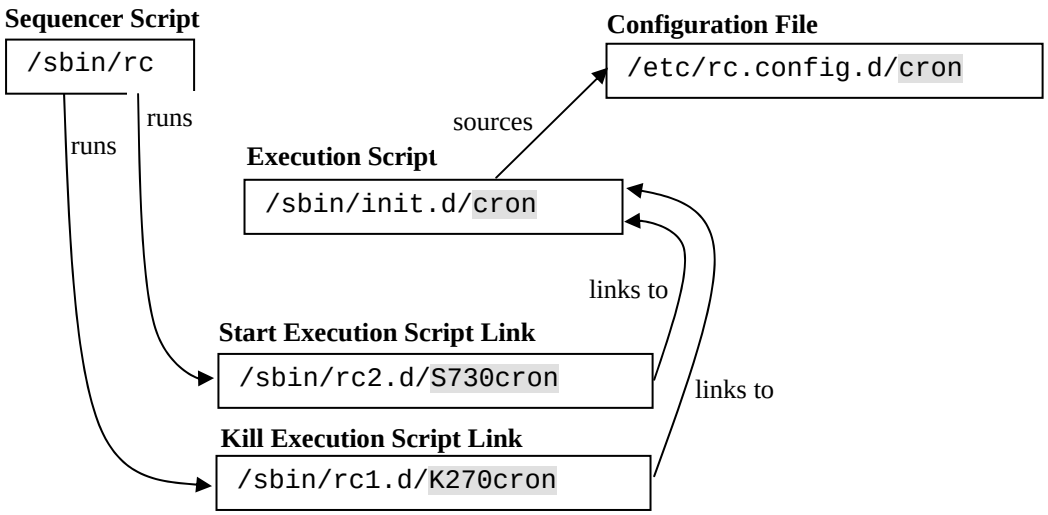
System Startup

INDEX

Introduction

The Run/Halt-Script based system startup that is used at HP-UX 10.X and 11.X follows the OSF/1 industry standard that has been adopted by Sun, SGI, and other vendors. There are four components involved: Execution Scripts, Configuration Files, Execution Script Links and Sequencer Script:

- **Execution Scripts** (`/sbin/init.d/*`)
  are located in `/sbin/init.d/`. They are invoked with either “start” or “stop” argument in order to start or stop the subsystem.
  Example: `/sbin/init.d/cron`
- **Configuration Files** (`/etc/rc.config.d/*`)
  are located in `/etc/rc.config.d/` and contain environment variables that are sourced by the execution scripts and control their behaviour. The naming convention implies that the name of a configuration file equals the name of the corresponding execution script.
  Example: `/etc/rc.config.d/cron`
- **Execution Script Links** (`/sbin/rc#.d/*`)
  are located in the `/sbin/rc`*n*`.d` directories and are named `S`*NNNscriptname*, where *NNN* is a sequence number, and *scriptname* is the name of an Execution script in the `/sbin/init.d` directory. Each of these scripts is actually a symbolic link to a run/halt script in `/sbin/init.d`. The `/sbin/rc` script runs them in order by sequence number.
  Example: `/sbin/rc2.d/S730cron` (a.k.a. Start Script Link)
  `/sbin/rc1.d/K270cron` (a.k.a. Kill Script Link)
- **Sequencer Script** (`/sbin/rc`)
  is started from the inittab and invokes each Execution script in the directories `/sbin/rc0.d`, `/sbin/rc1.d`, `/sbin/rc2.d`, `/sbin/rc3.d`, and `/sbin/rc4.d`, in that order. It sources all the files in the `/etc/rc.config.d/` directory.
- **Log file** (`/etc/rc.log`)
  is `/etc/rc.log`. The log file of the previous system start is `/etc/rc.log.old`



Configuration Files - /etc/rc.config.d/*

The system configuration used at startup is contained in files within the directory `/etc/rc.config.d`. The script `/etc/rc.config` sources all of the files within `/etc/rc.config.d` and `/etc/TIMEZONE` and exports their contents to the environment.

/etc/rc.config

The file `/etc/rc.config` is a script that sources all of the `/etc/rc.config.d/*` scripts, and also sources `/etc/TIMEZONE`. To read the configuration definitions, only this file need be sourced. This file is sourced by `/sbin/rc` whenever it is run, such as when the init command is run to transition between run states. Each file that exists in `/etc/rc.config.d` is sourced, without regard to which startup scripts are to be executed.

```
# cat /etc/rc.config

# This script sources configuration files for various subsystems. Configuration
# variables are in files CFG_FILE in /etc/rc.config.d/*, and /etc/TIMEZONE.
# Simple error checks are done: test file perms and for dumped "core" files.
# Filenames containing [.,~#] are not sourced - they are possible names of
# backup files such as file.bak, file.OLD, ~file, file~, file,v, #file etc.
#
# @(#)B.11.11_LR

for CFG_FILE in /etc/rc.config.d/* /etc/TIMEZONE
do
   fname=${CFG_FILE##*/}      # get file basename
   if [ -f $CFG_FILE -a "$fname" != "core" -a "${fname##*[.,~\#]}" = "$fname" ]
   then . $CFG_FILE        # source a valid config file
   fi
done
```

/etc/rc.config.d

The configuration information is structured as a directory of files, rather than as a single file containing the same information. This allows developers to create and manage their own configuration files here, without the complications of shared ownership and access of a common file.

NOTE: Do not store any other files but valid configuration files into this directory. It would be sourced by rc.config at startup and result in errors being logged to `/etc/rc.log`. It is not seldom that files named e.g. `ioscan.out` or `netconf.old` are left in this directory ;-)

/etc/rc.config.d/*

This is where files containing configuration variable assignments are located. Configuration scripts must be written to be read by the **POSIX shell**, and not the Bourne shell, ksh, or csh. In some cases, these files must also be read and possibly modified by Software Distributor control scripts or SAM. See `sd(4)` and `sam(1M)`. For this reason, each variable definition must appear on a separate line, with 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 one. This example shows the required syntax for configuration files:

```
# Cron configuration. See cron(1M)

# CRON: Set to 1 to start cron daemon
#
CRON=1
```

Configuration variables may be declared as array parameters when describing multiple instances of the variable configuration. For example, a system may contain two network interfaces, each having a unique IP address and subnet mask (see `ifconfig(1M)`). An example of such a declaration is as follows:

```
INTERFACE_NAME[0]=”lan1”
IP_ADDRESS[0]=”15.1.55.2”
SUBNET_MASK[0]=”255.255.248.0”

INTERFACE_NAME[1]=”lan3”
IP_ADDRESS[1]=”15.1.55.3”
SUBNET_MASK[1]=”255.255.248.0”
```

Note that 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. There is no protection against environment variable namespace collision in these configuration files. Programmers must take care to avoid such problems.

Its highly recommended that the name of a configuration file is the same as the name of the appropriate Execution Script at **`/sbin/init.d/`**.

/etc/TIMEZONE

The file `/etc/TIMEZONE` contains the definition of the `TZ` environment variable. This file is required by POSIX. It is sourced by `/sbin/rc` at the same time the `/etc/rc.config.d/*` files are sourced.

```
# cat /etc/TIMEZONE

TZ=MET-1METDST
export TZ
```

Execution Scripts - /sbin/init.d/*

Each of the scripts in `/sbin/init.d/` deals with one application or one subsystem, such as NFS server daemons. A single script in `/sbin/init.d/` must be able to start up an application (or subsystem) or kill it. The script will be passed the command line parameter 'start' or 'stop' depending on whether it is supposed to start or kill the appropriate daemons.

In some cases, only a start action will be applicable. If this is the case, and if the stop action is specified, the script should produce a usage message and exit with an error. In general, scripts should look at their arguments and produce error messages if bad arguments are present. When a script executes properly, it must exit with a return value of zero. If an error condition exists, the return value must be nonzero.

The modularity is important, i.e. there should be no script invoking another script. Modifying the scripts to change the behaviour is not permitted. This should be done by modifying the variables in the Configuration Files.

Arguments

The Execution scripts should be able to recognize the following four arguments (where applicable):

`start`	The start argument is passed to scripts whose names start with S. Upon receiving the start argument, the script should perform its start actions.
`stop`	The stop argument is passed to scripts whose names start with K. Upon receiving the stop argument, the script should perform its stop actions.
`start_msg`	the start_msg argument is passed to scripts whose names start with S so that 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 routines. Scripts given just the start_msg argument will only print a message and not perform any actions.
`stop_msg`	the stop_msg argument is passed to scripts whose names start with K 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. Scripts given just the stop_msg argument will only print a message and not perform any actions.

Script Output

To ensure proper reporting of startup events, startup scripts are required to comply with the following guidelines for script output.

- Status messages, such as "`starting house daemon`" must be directed to **stdout**. All

error messages must be directed to **stderr**.

- Script output, both stdout and stderr, is redirected to log file `/etc/rc.log`, unless the startup checklist mode is set to the raw mode. In this case, all output goes to the console. All error messages should be echoed to stdout or stderr.
- Startup scripts are not allowed to send messages directly to the console, or to start any daemons that immediately write to the console. This restriction exists because these scripts are now started by the `/sbin/rc` checklist wrapper. All script output should go to either stdout or stderr, and thus be captured in a log file. Any console output will be garbled.

Return Values

The return values for Execution 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 (NOTE: Reserved for key system components).
4	Script exited without error and started a process in background mode.
>4	For return values greater than 4 the action is same as return value 1, script encountered errors.

NOTE: 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!

Sequencer Script - /sbin/rc

The job of the sequencer script `/sbin/rc` is to invoke the individual startup scripts in `/sbin/init.d/`. However, `/sbin/rc` does not invoke these scripts directly, but rather through a series of symbolic links (Execution Script Links) in the `/sbin/rc?.d/` directories. For example, if your system is booting to run level 3 (invoking all services, including CDE), then `/sbin/rc` will invoke all the scripts (symbolic links) in the `/sbin/rc2.d/`, and `/sbin/rc3.d/` directories.

The names of these links in `/sbin/rc?d/` directories begin with a **K** or an **S**. `/sbin/rc` will invoke the ones that begin with a K using the 'stop' parameter and the ones that begin with an S using the 'start' parameter. `/sbin/rc` does not really care about the serial numbers in the link names after the K or S; they are there so that you can configure the order in which the scripts are run, since they run in alphanumeric order. If you don't care when your software starts, use serial number 900 for an S file and 100 for a K file, which are guaranteed not to conflict with existing serial numbers.

The `/sbin/rc` script is the one that interprets the exit codes from the script, so if you write your own script make sure to get the exit codes correct. At boot time, `/sbin/rc` creates a checklist on the console, filling in each box with OK, FAIL, or N/A corresponding to exit codes 0, 1, or 2 from the script. If the script exits with code 3 or something other than 0, 1, or 2, `/sbin/rc` could reboot the system. So don't use that exit code spuriously.

The Sequencer `/sbin/rc` is invoked upon entering a new run level via the init N command (where N equals 0-6). The script /sbin/rc is typically invoked by the corresponding entry in the file /etc/inittab as follows:

```
sqnc:123456:wait:/sbin/rc </dev/console >/dev/console 2>&1
```

If a transition from a lower to a higher run level (i.e., init state) occurs, the start scripts for the new run level and all intermediate levels between the old and new level are executed. If a transition from a higher to a lower run level occurs, the kill scripts for the new run level and all intermediate levels between the old and new level are executed.

If a start script link (e.g., `/sbin/rc2.d/S730cron`) in sequencer N has a stop action, the corresponding kill script should be placed in sequencer N-1 (e.g., `/sbin/rc1.d/K270cron`). Actions started in level N should be stopped in level N-1. This way, a system shutdown (e.g., transition from level 3 directly to level 0) will result in all subsystems being stopped.

Execution Script Links - /sbin/rc#.d/*

The Execution Script Links are located in the Sequencer (or Run Level) Directories `/sbin/rcN.d/` where N reflects the Run Level. The contents of sequencer directories consist of symbolic links to startup scripts in `/sbin/init.d`. These symbolic links must follow a strict naming convention, as noted in the various fields of this example:

`/sbin/rc2.d/S060cron`

`rc2.d` Sequencer (Run Level) Directory:
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 upon entering run-level 2 from runlevel 3.

`S` 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").

`060` 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. Although it is not recommended, two scripts may share the same sequence number.
Kill scripts for a start script in directory `/sbin/rcN.d` will reside in `/sbin/rc(N-1).d`. E.g. `/sbin/rc3.d/S499homer` and `/sbin/rc2.d/K501homer` might be execution counterparts.
By convention the sequence numbers of start and kill script sum up to 1000.

`cron` 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`.
NOTE: UX 10.X still supports short file names (14 or less characters). This means that the script name field and therefore the Executiopn SCript in `/sbin/init.d` is limited to 10 or fewer characters.

Scripts are executed in lexicographical order defined by the shell. 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. HP-supplied sequencer entries will all have unique numbers.

When ordering start and kill script links, note that subsystems started in any given order should be stopped in the reverse order to eliminate any dependencies between subsystems. This means that kill scripts will generally not have the same numbers as their start script counterparts. E.g. if two subsystems must be started in a given order due to dependencies

(e.g., S111house followed by S222uses_house), the kill counterparts to these scripts must be numbered so that the subsystems are stopped in the opposite order in which they were started (e.g., K778uses_house followed by K889house).

Run Level Definitions

When entering from lower state (run level), all start scripts are executed, and when entering from higher states, all kill scripts are executed. This is applicable for run levels 1 through 6.

The following table shows a typical system startup.

Run level	State	Sequencer Dir	subsystems that are started
0	Halted	`/sbin/rc0.d`	/
S	Single User	`/sbin/rc0.d`	/
1	Minimal	`/sbin/rc1.d`	configure system for crash dump mount local file systems set hostname enable secondary swap start syncer
2	Multi-User	`/sbin/rc2.d`	configure unconfigured software start syslog daemon configure lan interfaces startup network start NFS client (mount NFS filesystem) start cron start Measureware daemon
3	Exported	`/sbin/rc3.d`	start NFS server (export NFS filesystems) join ServiceGuard cluster if applicable This is the default run level
4	CDE	`/sbin/rc4.d`	start graphical user interface
5	Not used	n/a	customer applications
6	Not used	n/a	customer applications

Example

The file `/sbin/init.d/template` is a good starting place for making your own start/stop scripts. The example here uses some of the comments from that template.

In the example, a file /sbin/init.d/mygame will start or stop a program called `/opt/mygame/mygamed`, a multi-player daemon used by the engineering staff to maintain contact with each other during the day. The daemon should be started at boot time and stopped on shutdown.

The script reads the configuration file `/etc/rc.config.d/mygame`. It will start the daemon only if the `MYGAME` variable is set to `1`. It will then use the `MAXPLAYERS` variable as an argument to send to the daemon.

After putting these files in place, make symbolic links to the `/sbin/init.d/` script so that it gets run properly by `/sbin/rc`:

```
ln -s /sbin/init.d/mygame /sbin/rc2.d/S900mygame
ln -s /sbin/init.d/mygame /sbin/rc1.d/K100mygame
```

Here is the **`/etc/rc.config.d/mygame`** configuration file:

```
# set MYGAME to 0 to disable the game daemon
MYGAME=1
# default MAXPLAYERS is 5
MAXPLAYERS=17
```

Here is the **`/sbin/init.d/mygame`** script:

```
#!/sbin/sh

# This script will read in ('dot' in) the file /etc/rc.config.d/mygame
# and start the 'mygamed' program if the MYGAME variable is set to 1.
# It also reads the MAXPLAYERS variable from the config file and passes
# that as an argument to the daemon.

# Allowed exit values:
#       0 = success; causes OK to show up in checklist.
#       1 = failure; causes FAIL to show up in checklist.
#       2 = skip; causes N/A to show up in the checklist.
#           Use this value if execution of this script is overridden
#           by the use of a control variable, or if this script is not
#           appropriate to execute for some other reason.
#       3 = reboot; causes the system to be rebooted after execution.

# Input and output:
#       stdin is redirected from /dev/null
#       stdout and stderr are redirected to the /etc/rc.log file
#       during checklist mode, or to the console in raw mode.

PATH=/usr/sbin:/usr/bin:/sbin
export PATH
```

```
    # NOTE:
    #   If your script executes in run state 0 or state 1, then /usr might
    #   not be available.  Do not attempt to access commands or files in
    #   /usr unless your script executes in run state 2 or greater.  Other
    #   file systems typically not mounted until run state 2 include /var
    #   and /opt.

    rval=0


    # Function to kill the named process(es).
    # $1=<search pattern for your process>

    killproc() {
            pid=`ps -e | awk '$NF~/'"$1"'/ {print $1}'`
            if [ "X$pid" != "X" ]; then
                    if kill "$pid"; then
                            echo "$1 stopped"
                    else
                            rval=1
                            echo "Unable to stop $1"
                    fi
            fi
    }


    case $1 in
    'start_msg')
        # Emit a _short_ message relating to running this script with
        # the 'start' arg; this message appears as part of the checklist.
        echo "Starting the mygamed daemon"
        ;;

    'stop_msg')
        # Emit a _short_ message relating to running this script with
        # the 'stop' arg; this message appears as part of the checklist.
        echo "Stopping the mygamed daemon"
        ;;

    'start')
        # source the system configuration variables
        MAXPLAYERS=5    # set a default to pass to daemon
        if [ -f /etc/rc.config.d/mygame ] ; then
                . /etc/rc.config.d/mygame
        else
                echo "WARNING: /etc/rc.config.d/mygame defaults file
MISSING"
        fi

        # Check to see if this script is allowed to run...
        if [ "$MYGAME" != 1 ]; then
            rval=2
        else
            # Execute the commands to start your subsystem
            /opt/mygame/mygamed -m $MAXPLAYERS
        fi
        ;;

    'stop')
        # source the system configuration variables
        killproc mygamed
        ;;

    *)
        echo "Usage: $0 {start|stop|start_msg|stop_msg}"
        rval=1
        ;;
    esac
```

```
exit $rval
# end of script
```

Troubleshooting

Many problems during system startup are caused by temporary files that have been left in the directory for configuration files (`/etc/rc.config.d/`). When files such as `ioscan.out`, `swlist.out` or `cron.bk` are sourced this can result in errors and even render a system unbootable. Only valid Posix shell scripts are allowed to be located in `/etc/rc.config.d/`.

An explanation on how to troubleshoot this problem is given at http://hprtnt06/mcux/MC-News/menue.html (HP internal).

Additional Information

Manual Pages

`rc(1M)`, `rc.config(4)`, `init(1M)`, `shutdown(1M)`, `inittab(4)`