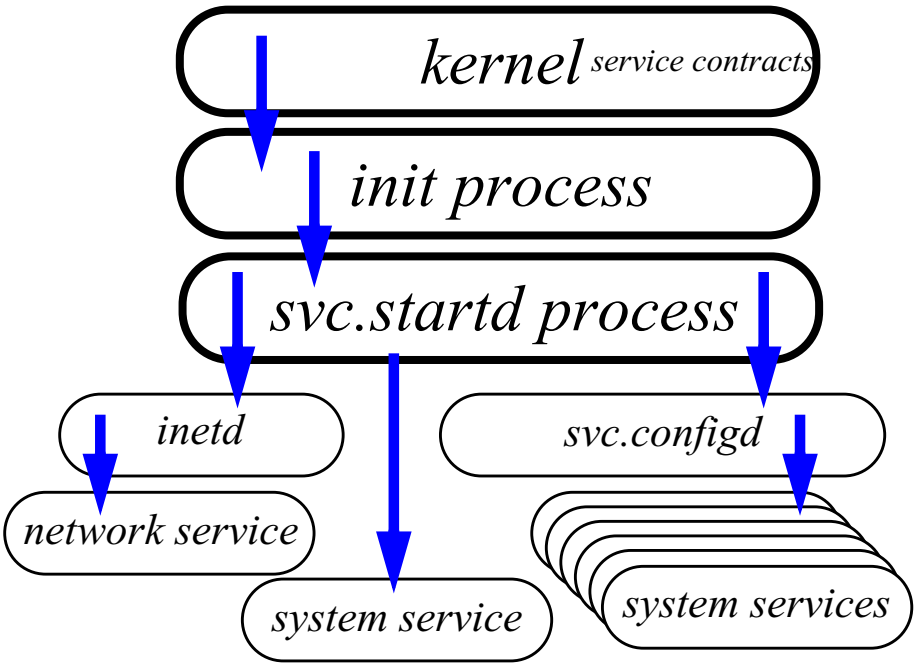
The Solaris 10 Service Framework

In earlier versions of Solaris, the *daemon* processes were started at boot time using shell scripts that ran in a pre-determined sequence. As such, all of the required *daemon* processes would be started but they would be running in isolation.

Because of this isolationist approach, it was possible that required *daemon* processes may not have started as they depended on another process being present.

To remedy this situation, Sun Microsystems have altered the mechanism used to manage the system's *daemon* processes so that the inter-dependencies are identified and managed. This can best be shown using the diagram below:



If a required process fails or is terminated, all dependant processes will be taken off-line until the required process is restarted in a stable state.

This inter-dependency is started by means of a *service contract*, maintained by the kernel, in which the process inter-dependency, the re-starter process and the start method are all described.

Rather than use the earlier Solaris release method of shell scripts to stop and start *daemon* processes, in the Solaris 10 release, they are now managed using service-related commands:

- `svcs` To display service related information
- `svcadm` To administer services
- `svccfg` To configure services
- `svcprop` To manage service properties
- `inetadm` To manage internet-daemon related services

Current services can be displayed as shown below:

```
# svcs
STATE          STIME    FMRI
legacy_run      7:38:56 lrc:/etc/rcS_d/S29wrsmcfg
legacy_run      7:39:18 lrc:/etc/rc2_d/S10lu
...
online          7:39:19 svc:/network/ssh:default
online          7:39:22 svc:/milestone/multi-user:default
online          7:39:29 svc:/milestone/multi-user-server:default
online          7:39:29 svc:/system/zones:default
offline         7:38:51 svc:/application/print/ipp-listener:default
offline         7:39:12 svc:/application/print/rfc1179:default
```

Legacy_run services are those that are still controlled via shell scripts.

Solaris 10 services can be in an `online` or `offline` state.

Services are managed by means of either:

- A service contract
- A service manifest

Where a contract exists, the service will be re-started if it has not been stopped using a service-related command.

The service manifest provides the detail of the service (*see page 3-22*).

Service contracts can be monitored using the following commands:

- `ctstat` To display service related information
- `ctwatch` To administer services

Services and Service Dependencies

What is a service?

A service is:

- An abstracted description of a single or group of processes that collectively provide a level of functionality
- Each service is clearly defined to the system, has a defined functional state and has defined error boundaries (*by means of the service contract*)
- A service has clearly defined methods and dependencies for starting, stopping, refreshing etc.

A process may invoke sub-processes; may require hardware resource and be subject to processing error states and hardware error states. How can the process continue running if error states occur and what happens to its sub-processes?

Each service is recognized by its Fault Management Resource Identifier (FMRI). The FMRI for any service would be unique amongst the services defined on that system. FMRIs can be either legacy services, for backwards-compatibility, or services:

```
legacy_run     12:28:07 lrc:/etc/rc2_d/S99dtlogin
disabled       12:26:39 svc:/network/dhcp-server:default
online         12:28:13 svc:/milestone/multi-user-server:default
maintenance    12:28:02 svc:/system/mdmonitor:default
```

Services can be in one of a number of states.

What is a contract?

A contract binds all of these together and forms a management structure that allows the system kernel and the processing controls to manage processes more effectively.

An extreme example could be - A process is abnormally terminated for some reason. The contract would allow the kernel and the management routines to re-start that process, even re-linking it with dependent sub-processes, once it has failed in an non-contractual manner.

The most important service-related command is the `svcs` command.

The svcs command

In order to see which services are currently running on a system, use the following command:

```
# svcs -a
STATE          STIME    FMRI
legacy_run     15:08:02 lrc:/etc/rcS_d/S50sk98sol
legacy_run     15:08:24 lrc:/etc/rc2_d/S10lu
legacy_run     15:08:25 lrc:/etc/rc2_d/S20sysetup
legacy_run     15:08:25 lrc:/etc/rc2_d/S40llc2
l...
disabled       15:07:46 svc:/system/metainit:default
disabled       15:07:48 svc:/network/rpc/keyserv:default
disabled       15:07:48 svc:/network/rpc/nisplus:default
disabled       15:07:48 svc:/network/nis/client:default
...
online         15:07:45 svc:/system/svc/restarter:default
online         15:07:49 svc:/network/pfil:default
online         15:07:50 svc:/network/loopback:default
online         15:07:50 svc:/milestone/name-services:default
...
online         15:08:28 svc:/milestone/multi-user:default
online         15:08:33 svc:/milestone/multi-user-server:default
online         15:08:36 svc:/system/zones:default
offline        15:08:19 svc:/application/print/rfc1179:default
```

To see which processes a service is dependent upon, use the -d option:

```
# svcs -d  cron
STATE          STIME    FMRI
online         12:26:41 svc:/milestone/name-services:default
online         12:27:10 svc:/system/filesystem/local:default
```

To see which processes are dependent upon a service, use the -D option:

```
# svcs -D  cron
STATE          STIME    FMRI
online         12:28:07 svc:/milestone/multi-user:default
```

If a service has failed for some reason and can not be started, you can list the service using the following command:

```
# svcs -x
svc:/system/mdmonitor:default (SVM monitor)
 State: maintenance since Tue 23 Aug 2005 12:28:02 PM BST
Reason: Start method failed repeatedly, last exited with status 1.
   See: http://sun.com/msg/SMF-8000-KS
   See: mdmonitord(1M)
   See: /var/svc/log/system-mdmonitor:default.log
Impact: This service is not running.
```

To list detailed information about a particular service, use the -l option:

```
# svcs -l cron
fmri         svc:/system/cron:default
name         clock daemon (cron)
enabled      true
state        online
next_state   none
state_time   Tue 23 Aug 2005 12:27:11 PM BST
logfile      /var/svc/log/system-cron:default.log
restarter    svc:/system/svc/restarter:default
contract_id  31
dependency   require_all/none svc:/system/filesystem/local (online)
dependency   require_all/none svc:/milestone/name-services (online)
```

From this information, we can see that the contract ID number is 31. Using a combination of programs we can ascertain which process in acting as a manager for the cron process:

```
# ctstat -i 31
CTID    ZONEID  TYPE    STATE   HOLDER  EVENTS  QTIME   NTIME
31      0       process owned   7       0       -       -
```

From this information, we can see that the HOLDER is PID number 7.

```
# ps -el | nawk 'NR == 1 ; $4 == 7'
 F S    UID   PID  PPID   C PRI NI     ADDR     SZ    WCHAN TTY         TIME CMD
 0 S      0     7     1   0  40 20        ?   1304        ? ?           0:08 svc.star
```

We can see that the cron process is being managed by the svc.startd process.

To determine which processes are related to a service, use the -p option;

```
# svcs -p svc:/application/print/server:default
STATE          STIME    FMRI
online         12:27:43 svc:/application/print/server:default
               12:27:43      329 lpsched

# svcs -p nfs/server
STATE          STIME    FMRI
online         14:32:34 svc:/network/nfs/server:default
               14:32:33     1661 mountd
               14:32:34     1665 nfsd
```

The SMF Identifiers

As we have seen from the previous listings, the FMRI will start with either `lrc` or `svc`.

For those whose FMRI starts with `svc`, the functional categories are:

- application
- device
- milestone
- network
- platform
- site
- system

By referring to the functional category, you can determine the area of work that the service is targeted at.

```
disabled       12:26:47 svc:/application/management/webmin:default

online         12:27:00 svc:/system/device/local:default

online         12:26:41 svc:/milestone/name-services:default

disabled       12:26:32 svc:/network/ipfilter:default

disabled       12:26:31 svc:/platform/sun4u/mpxio-upgrade:default

disabled       12:26:38 svc:/system/rcap:default
```

Identifiers can be complete or partial, so long as the service management controls can uniquely identify the FMRI:

```
# svcs svc:/milestone/name-services:default
STATE          STIME    FMRI
online         12:26:41 svc:/milestone/name-services:default
# svcs milestone/name-services
STATE          STIME    FMRI
online         12:26:41 svc:/milestone/name-services:default
# svcs name-services
STATE          STIME    FMRI
online         12:26:41 svc:/milestone/name-services:default
# svcs milestone
svcs: Pattern 'milestone' doesn't match any instances
STATE          STIME    FMRI
```

The svcadm command

Services should be stopped, refreshed and started using the svcadm command.

Note – The exceptions to this rule are legacy services, which should be stopped and started using the legacy run-control script and the Internet-related processes that must be managed using the inetadm command.

Services can be in one of the following states:

- degraded - the service instance in running but at a limited capacity
- disabled - the service instance is not enabled and not running
- legacy_run - the service instance can not be observed or managed by the SMF controls
- maintenance - the service has encountered an error state. To re-start this service, manual intervention is required.
- online - the service instance in enabled and started
- offline - the service instance is enabled but stopped
- uninitialized - the initial state for all services prior to their initialisation. For example, when the service manifest is read at boot-time

Shown below is an example of stopping and re-starting a service:

```
# svcs nfs/server
STATE          STIME    FMRI
online         14:32:34 svc:/network/nfs/server:default

# svcadm disable nfs/server

# svcs -p nfs/server
STATE          STIME    FMRI
disabled       14:53:08 svc:/network/nfs/server:default
```

```
# svcadm enable nfs/server

# svcs -p nfs/server
STATE          STIME    FMRI
online         14:54:25 svc:/network/nfs/server:default
               14:54:24     1750 mountd
               14:54:24     1752 nfsd

# svcadm -v restart nfs/server
Action restart set for svc:/network/nfs/server:default.

# svcs -p nfs/server
STATE          STIME    FMRI
online         14:55:50 svc:/network/nfs/server:default
               14:55:49     1803 mountd
               14:55:50     1805 nfsd
```

We can see from the differing PID numbers for the `mountd` and `nfsd` processes that they must have been stopped, completely, and then started, as brand-new processes, when the `nfs/server` service was restarted.

Services are configured using XML control files, referred to as service manifests.

The system combines all of the required service manifests into a single system registry. This registry is created at the time of the Solaris OS installation. However, as changes are made to the configuration of services, the system registry will be modified automatically.

```
# more /var/svc/manifest/network/nfs/server.xml
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<!--
        Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
        Use is subject to license terms.

        ident   "@(#)server.xml 1.10    04/12/16 SMI"

        NOTE:  This service manifest is not editable; its contents will
        be overwritten by package or patch operations, including
        operating system upgrade.  Make customizations in a different
        file.

        Note: if this service is modified to consist of anything other
        than a single instance named 'default', you must make changes to
        $SRC/head/rpcsvc/daemon_utils.h and libnsl:open_daemon_lock().
-->

<service_bundle type='manifest' name='SUNWnfssr:nfs-server'>
```

```
<service
        name='network/nfs/server'
        type='service'
        version='1'>

        <create_default_instance enabled='false' />

        <single_instance />

        <dependency name='network'
            grouping='require_any'
            restart_on='error'
--More--(27%)
```

Note – At the time of the first release of the Solaris 10 OS, the only mechanism provided to allow the creation of new service manifest is a manifest template. In future releases, it is expected that there will be GUI-based tools to aid in the building of manifests for custom services.

The svccfg command

The svccfg command allows the structure and properties of a service to be configured or displayed.

When invoked, you must supply an FRMI as an argument.

By default, svccfg will use the current repository view of the service as the basis of information. It is possible, however, to specify an alternative file that must be used.

Note – The system repository is a database of all current service details.

If properties are altered, they are altered in the current system repository by default.

To list properties of a service, use the following as an example:

```
# svccfg -v -s cron
svc:/system/cron> listprop
usr                           dependency
usr/entities                  fmri     svc:/system/filesystem/local
usr/grouping                  astring  require_all
usr/restart_on                astring  none
usr/type                      astring  service
ns                            dependency
ns/entities                   fmri     svc:/milestone/name-services
ns/grouping                   astring  require_all
ns/restart_on                 astring  none
ns/type                       astring  service
general                       framework
general/action_authorization  astring  solaris.smf.manage.cron
general/entity_stability      astring  Unstable
general/single_instance       boolean  true
dependents                    framework
dependents/cron_multi-user    astring  svc:/milestone/multi-user
startd                        framework
startd/ignore_error           astring  core,signal
start                         method
start/exec                    astring  /lib/svc/method/svc-cron
start/group                   astring  root
start/limit_privileges        astring  :default
start/privileges              astring  :default
start/project                 astring  :default
start/resource_pool           astring  :default
start/supp_groups             astring  :default
start/timeout_seconds         count    60
...
```

```
stop                          method
stop/exec                     astring  :kill
stop/timeout_seconds          count    60
stop/type                     astring  method
tm_common_name                template
tm_common_name/C              ustring  "clock daemon (cron)"
tm_man_cron                   template
tm_man_cron/manpath           astring  /usr/share/man
tm_man_cron/section           astring  1M
tm_man_cron/title             astring  cron
tm_man_crontab                template
tm_man_crontab/manpath        astring  /usr/share/man
tm_man_crontab/section        astring  1
tm_man_crontab/title          astring  crontab
svc:/system/cron> quit
```

From this list of properties, we can see that the `start/exec` method is to make use of the file called `/lib/svc/method/svc-cron`. On closer investigation, we find that this is a shell script:

```
# cat /lib/svc/method/svc-cron
#!/sbin/sh
#
# Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)svc-cron   1.2     04/11/05 SMI"
#
# Start method script for the cron service.
#

. /lib/svc/share/smf_include.sh

if [ -p /etc/cron.d/FIFO ]; then
        if /usr/bin/pgrep -x -u 0 -z ‘/sbin/zonename‘ cron >/dev/null 2>&1; then
                echo "$0: cron is already running"
                exit $SMF_EXIT_ERR_NOSMF
        fi
fi

if [ -x /usr/sbin/cron ]; then
        /usr/bin/rm -f /etc/cron.d/FIFO
        /usr/sbin/cron &
else
        exit 1
fi
exit $SMF_EXIT_OK
```

Ultimately, all services will either start or stop processes as the services are started or stopped.

Milestones

When considering the Service Management Facility (SMF) it is vital that we understand the concept of *milestones*.

A milestone is a given state of the system.

For example, if the system is currently at the single-user milestone state, there will be very few processes running on the system and users will not be able to log on to the system from remote terminals or networked hosts.

If the system is currently at the multi-user milestone state, there will be a number of client processes running but few or no server processes running, even though the system may normally act as a server.

By changing the milestone state, you are stopping or starting all the relevant processes that will be required to enter that state.

```
# svcs -a | grep milestone
online         12:26:41 svc:/milestone/name-services:default
online         12:26:49 svc:/milestone/network:default
online         12:27:05 svc:/milestone/devices:default
online         12:27:09 svc:/milestone/single-user:default
online         12:27:19 svc:/milestone/sysconfig:default
online         12:28:07 svc:/milestone/multi-user:default
online         12:28:13 svc:/milestone/multi-user-server:default
```

How to determine the current milestone state

In order to know whether or not processes should be running, you need to know the current milestone state of the system.

To determine the system default state, use the following command:

```
# svcprop svc:/system/svc/restarter:default  | grep milestone
```

If no output is produced, you can assume that the default milestone is `all`, which is equivalent having all configured milestones functional. The `all` milestone is not the same as the `multi-user-server` milestone.

Worked Example: Milestone Changes

Here is a worked example of changing the running processes by altering the current milestone.

Note – If a milestone is set using the `svcadm` command without the `-d` option, the milestone change takes place but the system default remains at its previous state. By using the `-d` option, you are both changing the current state and setting a new default for the system at next boot-time.

1. First, we set the current milestone to multi-user. Because the `-d` option is used, this will also become the current system default state:

```
# svcadm milestone -d multi-user
```

2. Next, we view the service properties to see what the default milestone is set to:

```
# svcprop svc:/system/svc/restarter:default | grep milestone
options/milestone astring svc:/milestone/multi-user:default
```

3. We also want to see just how many processes are currently running, so we issue the following command:

```
# ps -e | wc -l
      48
```

4. Now, we set the default milestone (*and the current milestone*) to `all`:

```
# svcadm milestone -d all
```

5. We view the service properties to see what the default milestone is set to:

```
# svcprop svc:/system/svc/restarter:default | grep milestone
options/milestone astring all
```

6. Again, we check to see how many processes are currently running:

```
# ps -e | wc -l
      61
```

7. Next, we set the current milestone to multi-user-server:

```
# svcadm milestone multi-user-server
```

8. Now, we view the service properties to see what the default milestone is set to see whether the previous command has changed the default:

```
# svcprop svc:/system/svc/restarter:default | grep milestone
options/milestone astring svc:/milestone/multi-user:default
options/milestone astring all
```

Note – If two milestone entries are displayed, this means that your system has a default milestone that is different from the current milestone state.

9. Once more, we check the number of running processes:

```
# ps -e | wc -l
      55
```

10. We set the milestone back to `all` and, just to be certain, also set the default level to be `all`:

```
# svcadm milestone -d all
```

11. We view the service properties to see what the default milestone is set to:

```
# svcprop svc:/system/svc/restarter:default | grep milestone
options/milestone astring all
```

Note – Now that we see only one milestone value, we can be certain that our system is running at the default milestone state.