

EnterpriseSMS Startup Scripts

A. General Information

The EnterpriseSMS startup scripts run on initial system boot and when the utility team_start [scriptname] is executed. They contain line entries for each of the programs (processes) that collectively constitute EnterpriseSMS. These processes are shut down automatically when the system is shut down, or when the utility team_stop [scriptname] is executed.

On UNIX, the startup scripts must be edited to add support for any new site, SIM, MonSIM, OCS, IVS, AMS, or SMS. Other system modifications also call for script editing to be performed.

When editing scripts, avoid the use of the TAB key when creating spaces between entries. The TAB key is interpreted as a system character, and can cause script execution to fail.

1. Execution Sequence

When the server boots, the AIX startup script /etc/inittab calls two other scripts that start the EnterpriseSMS processes. In SCO UNIX, this script is S99zstart. The script rc.oracle starts the Oracle® system processes; the script rc.amm starts the specific control processes by calling other scripts:

- ccm_mon - Starts the Command and Control Module processes for redundant server configurations.
- redun_procs - starts the redundant processes in multiple server configurations.
- amm_mon - Starts the Command and Control Module processes for standalone server systems.
- nim_mon - Starts the Subsystem Interface Module processes.

Station startup files may then be started manually. These scripts include:

- ocsname_mon - Starts the Operator Command Station support processes.
- smsname_mon - Starts the Status Monitor Station support processes.
- monsim_mon - Starts the Monitor SIM support processes (example not shown).
- ivsname_mon - Starts the Image Verification Station support processes.
- amsname_mon - Starts the Activity Monitor Station support processes.

2. Script Structure

Each line in the scripts starts a specific program, and is entered into the file in a standard format. This format is as follows:

```
alias:start mode:executable_name [optional arguments] >.out file
```

The *alias* is the name by which UNIX will log the executable. If the running processes are viewed using either the team_lui or ps -ef command, the output will display the alias of the executable, not the name of the executable itself.

The *Start Mode* determines how UNIX manages the process. Three start modes are supported:

- Off. The process does not start.
- Wait. The process must complete before subsequent processes load.
- Respawn. The process loads and runs continuously. Most EnterpriseSMS processes are run in respawn mode. If a process in respawn ends abnormally, UNIX notes the event and restarts the process automatically.

The *Executable Name* is the actual name of the program started by the script entry. The executable must reside in the same directory as the script starting the file. If the executable is not present in that

directory, or does not exist, the following message will display and post to the hostname.log file located in that directory:

```
Unable to find all processes
```

The `.outfile` is the name of the file, in the format `alias.out`, that is created by the `system` to log output data from process when it is placed in debug mode.

3. Arguments to Program Startup

Each program started in the `_mon` files has a number of required and/or optional arguments which affect its execution. These arguments are documented along with the script entries later in this chapter.

One argument which is used globally is the `_d` argument. It is used to place the program process in *debug* mode for troubleshooting purposes. When in debug, an output log is generated and saved using the filename found after the `_` sign in the script entry. Users may view these `.out` files using the UNIX `more` command.

Processes should not be left in debug without a specific reason ; the use of debug may slow system processes. The utility `team_lui` , which displays system processes, will annotate which processes are in debug when run.

Also, `.out` files created by debug can become very large; they should be deleted after the problem is rectified. This may be done either with the UNIX `rm` command, or through use of the `z_clean` utility.

4. Comparison: Redundant vs. Single Server Entities

In a single server configuration, `amm_mon` executes first. On a redundant system, the `ccm_mon` file executes, starting the redundant processes. A copy of the `start_stop_px` process starts for each server in the redundant entity. It then calls the scripts `redun_procs` and `amm_mon`.

Unlike the majority of the standard processes, not all redundant processes respawn. Some start and stop, others respawn depending on the state of the server. When a server needs to run certain redundant processes, it causes the proper script to run, starting those processes.

The redundant servers are named in the files `/usr/ocs/hosts.txt` and `/usr/sim/bin/hosts.txt`. These server names are then defined in the `start_stop_px` process.

Because of this, the startup scripts should contain no `-h[server name]` arguments. The omission of these arguments causes the system to search `hosts.txt` file and attempts to contact each server found in the file sequentially until a response is received.

Entries in `hosts.txt` file are in the format (where `#` is the number of servers in the redundant entity):

```
host1 (system tcp/ip alias of host 1)
host2 (system tcp/ip alias of host 2)
host(...) (system tcp/ip alias of host ...)
```

`amm_mon` should never be run as an independent process in a redundant entity.

If `ccm_mon` terminates, it attempts to kill all redundant processes with the exception of `start_stop_px`. It then executes the `team_stop amm` command.

ccm_mon

Concept

Each server in a redundant entity runs its own `ccm_mon` and `redun_procs` script. These scripts are identical with one another, with the exception of the `-h` option (each calling the opposite server) and the `-p` parameter which identifies the `ccm_main` process running on the primary server.

Script Entries

```
ccm_main:once:ccm_main > ccm_main.out
```

Starts redundant processing; establishes communications between the servers.

- -h x. When running on the primary server, x is the alias of the secondary server; when running on the secondary server, x is the name of the primary server.
- -p defines the ccm_main module as running on the primary server.

start_stop_p1:respawn:start_stop_p > start_stop_p.out

Establishes redundant message queues; messages in start_stop_p_q contain commands to start and stop certain redundancy processes.

- -p x, where x is the queue name (typically start_stop_p_q, where q designates the number of servers in the redundant entity. Contains commands for starting and stopping redundant processes.
- start_stop_p2:respawn:start_stop_p > start_stop_p.out
- -a x, where x is the queue name (typically start_stop_amm_q, where q designates the number of servers in the redundant entity. Contains the commands for starting and stopping certain redundant processes.

ims_send:respawn:ims_send > ims_send.out

Transfers images between redundant servers. Runs on the master server (host/subhost configurations) or on both servers (multi-master configuration).

- -t ##, where ## is the number of seconds between updates.
- -s x where x is the name of the service located in /etc/services (typically, ims_serv?), where ? is the number of the servers in the redundant entity cited in the hosts.txt file. When referring to the primary server, the ? is omitted).

ims_rcv:respawn:ims_rcv > ims_rcv.out

Transfers images between redundant servers. Runs on the subhost (host/subhost configurations) or on both servers (multi-master configurations).

- -t ##, where ## represents the number of seconds between updates.
- -s x, where x is the name of the service located in /etc/services (typically, ims_serv?, where ? is the number of servers in the redundant entity cited in the hosts.txt file. When referring to the primary server, ? is omitted).

Example Script

```
ccm_main:once:ccm_main -h host2 -p > ccm_main.out
start_stop_p1:respawn:start_stop_p -p > start_stop_p.out
start_stop_p2:respawn:start_stop_p -a > start_stop_amm.out
ims_send:respawn:ims_send -t 60 -s ims_serv > ims_send.out
ims_rcv:respawn:ims_rcv -s ims_serv2 > ims_rcv.out
```

redun_procs

Script Entries

ping_backup:ping_backup > ping_backup.out

Runs on the primary server; constantly verifies communication between the servers.

-h x, where x is the name of the opposite server in the redundant entity.

rcv_ping:rcv_ping > rcv_ping.out

Runs on the secondary server; constantly verifies communication between the servers.

-h x, where x is the name of the opposite server in the redundant entity.

watch_host:watch_host > watch.out

Reports if the server is running.

-h x, where x is the name of the opposite server in the redundant entity.

send_upd:send_upd > send_upd.out

Sends update messages from one server to the other.

-h x, where x is the name of the opposite server in the redundant entity.

recv_upd:recv_upd > recv_upd.out

Reads the update messages from one server to the other.

sync_recv:sync_recv > sync_recv.out

Receives messages and data from the sync_send process on the other server; updates its own data tables.

sync_send:sync_send > sync_send.out

Reads all records that need to be synchronized between the servers and sends them to the sync_recv process on the other server.

-h x, where x is the name of the opposite server in the redundant entity.

ora_init:ora_init > ora_init.out

Executes the Oracle® initialization script. Retrieves ccm-related data from the database when the CCM is started.

Example Script

```
ping_backup:ping_backup -h host2 > ping_backup.out
recv_ping:recv_ping -h host2 > recv_ping.out
watch_host:watch_host -h host2 > watch.out
send_upd:send_upd -h host2 > send_upd.out
recv_upd:recv_upd > recv_upd.out
sync_recv:sync_recv > sync_recv.out
sync_send:sync_send -h host2 > sync_send.out
ora_init:ora_init > ora_init.out
```

amm_mon

Concept

There are two possible configurations for OCS's, SMSs, AMSs, and IVSs: Client/Server, and X-Terminal. The configurations are mutually exclusive; a given server can connect either client/server or X-Terminal, not both.

In a client/server configuration, each station must have an udp_send process with the -a argument. In an X-terminal configuration, the udp_send -a processes are replaced by a single aws_send process. The aws_send process executable is located in the /usr/aws directory; all other processes are located in the /usr/amm/process directory.

Script Entries

ora_init:wait:ora_init > ora_init.out

Runs the Oracle® startup scripts. Retrieves ccm-related data from the database when the CCM is

started.

alm_proc:respawn:alm_proc > alm_proc.out

Receives information about alarm conditions from other processes in the system and routes them to the appropriate OCS.

-r indicates to the ora_intrf process that database redundancy is on.

alm_servr:respawn:alm_servr > alm_servr.out

Services requests from the Operator Command Stations.

-r indicates to the ora_intrf process that database redundancy is on.

amag_dbase:respawn:amag_dbase > amag_dbase.out

Passes data from Oracle to the SIM.

cctv_dummy:respawn:cctv_dummy > cctv_dummy.out

Mutually exclusive with the cctv_if process.

cctv_if:off:cctv_if > cctv_if.out

-t terminal server identifies the terminal server. Valid are xyplex or lantronix.

-s service identifies the service. Vicon, ad, javelin, or burle are valid.

-v technology identifies the technology type. Vicon, ad, javelin, or burle are valid.

cmd_interp:respawn:cmd_interp > cmd_interp.out

Creates the data path for CCTV and/or intercom connections.

cmd_proc:respawn:cmd_proc > cmd_proc.out

Processes event requests (Event Responses and/or Guard Tour) from the OCS.

gtour_proc:respawn:gtour_proc > gtour_proc.out

Monitors guard tours in progress.

icom_if:respawn:icom_if -dummy > icom_if.out

Interface process to the intercom system.

-dummy is used to clear process queues as the process runs (required).

inc_update:respawn:inc_update > inc_update.out

Monitors the database for updates to devices, events, and alarm information.

-i# is the interval, in minutes, between which the process checks Oracle® for database updates. Default is 1 minute . Large installations may increase throughput by increasing the number of minutes between checks.

log_servr:respawn:log_servr > log_servr.out

Logs alarms and other system activities to the event log (event_log_tbl) and routes alarms for printing.

-no_print turns off automatic alarm printing.

logtbl_proc:respawn:logtbl_proc -e 50000 -a 100000 > logtbl_proc.out

Checks the Oracle® log tables once daily; generates alarm messages if greater than 70%, 80%, and 90% full.

-e sets the 100% full value for the event log (in rows). Default is 50000 records.

-a sets the 100% full value for the archive log (in rows). Default is 100000 records.

useorlose:respawn:useorlose X > useorlose.out

X determines the number of days a badge may be unused before automatic inactivation. Default is 10 days . Badges not used during that period have their badge status changed to Inactive in the database badge_tbl. Inactivations post to the event log with an event code of Lose. Valid are 1 to 32767 days.

If used, the system builds bdglast files in the /usr/amm_data_files directory. These files should be backed up independently from the database when the system is backed up, and restored independently if the database is restored from backup. If the bdglast files are erased or not restored after system loss while userorlose is on , the system will set all badges to Inactive on boot.

ora_intrf:respawn:ora_intrf > ora_intrf.out

Retrieves CCM-related data from the database (login validation, badge query lookup, etc.)

-r is required when database redundancy is on.

ping_aws:respawn:ping_aws > ping_aws.out

Continuously verifies communications between server and clients.

-i ## represents the interval between verifications (## in seconds).

recv_serv:respawn:recv_serv > recv_serv.out

Causes data download to the SIM after the SIM restarts.

aws_info:respawn:aws_info > aws_info.out

The primary data interface between the CCM and the OCS processes.

ims_send:off:ims_send > ims_send.out

Transfers stored badgeholder images between redundant servers. Normally off in single server configurations. In a redundant entity, runs on the master server.

-t ## represents the number of seconds (##) between server updates.

ims_rcv:off:ims_rcv > ims_rcv.out

Transfers stored badgeholder images between redundant servers. Runs on the secondary server. Normally off in single server configurations.

-t ## represents the number of seconds (##) between updates.

udp_rcv:respawn:udp_rcv > udp_rcv.out

Receives alarm information from the SIM.

rts_lib:respawn:rts_lib > rts_lib.out

Statuses devices for the Status Monitor Station (optional feature).

metasys_if:respawn:metasys_if > rts_lib.out

Supports the optional Metasys interface. Optional Feature.

-w x, where x is the Metasys workstation name.

-u y, where y is the Metasys user name.

-p ?, where ? is the Metasys password.

Required Entries for SIMs (client/server configuration)

udp_send??:respawn:udp_send > udp_send.HOST_DBNAME.out

Provides interface support to the SIM. ?? represents any number, usually sequential, used to make the alias unique.

- n defines the process as a SIM process.
- w OCSname where the OCSname is the same as that entered into the DCS DEVICES-NETWORK STATIONS-OCSs form.
- i ## sets the message interval. The server must acknowledge message receipt within this time; if not acknowledged, a timeout will occur, initiating a resend. 1-99 seconds are valid. Default is 3.
- e ## is the number of times a message will resend before an alarm is logged. Default is 3.

nim_dload??:respawn:nim_dload > nim_dload.SIM_DBNAME.out

Provides interface support to the SIM. ?? represents any number, usually sequential, used to make the alias unique.

- t servername where servername is the name of the computer on which the SIM runs.
- s simname is the name assigned in the Oracle® database.

Required Entries for a MonSIM (client/server configuration)

udp_send??:respawn:udp_send > udp_send.MONSIM_DBNAME.out

Provides interface support to the MonSIM. ?? represents any number, usually sequential, used to make the alias unique.

- n defines the process as a MonSIM process.
- t target where target is the defined alias of the client system that will run the external SIM, MonSIM, SMS, AMS, OCS, or IVS.
- w targetname where targetname is the name of the external SIM, MonSIM, or station as assigned in the Oracle® database.
- i ## sets the interval value. The server must acknowledge message receipt within this time; if not acknowledged, a timeout will occur, initiating a resend. 1-99 seconds are valid. Default is 2 seconds.
- e ## is the number of times a message will resend before a "device off-line" alarm is logged. Default is 5.

Required Entries for a Status Monitor Station (client/server configuration)

udp_send??:respawn:udp_send > udp_send.SMS_DBNAME.out

Provides interface support to the Status Monitor Station. ?? represents any number, usually sequential, used to make the alias unique.

- s defines this process as a Status Monitor Station process.
- t target where target is the defined alias of the client system that will run the external SIM, MonSIM, SMS, AMS, OCS, or IVS.
- w targetname where targetname is the name of the external SIM, MonSIM, or station as assigned in the Oracle® database.
- i ## sets the interval value. The server must acknowledge message receipt within this time; if not acknowledged, a timeout will occur, initiating a resend. 1-99 seconds are valid. Default is 2 seconds.

-e ## is the number of times a message will resend before a "device off-line" alarm is logged.
Default is 5.

Required Entries for an Activity Monitor Station (client/server configuration)

udp_send?:respawn:udp_send > udp_send.AMS_DBNAME.out

-m defines this process as an Activity Monitor Station process.

-t target where target is the defined alias of the client system that will run the external SIM, MonSIM, SMS, AMS, OCS, or IVS.

-w targetname where targetname is the name of the external SIM, MonSIM, or station as assigned in the Oracle® database.

-i ## sets the interval value. The server must acknowledge message receipt within this time; if not acknowledged, a timeout will occur, initiating a resend. 1-99 seconds are valid. Default is 2 seconds.

-e ## is the number of times a message will resend before a "device off-line" alarm is logged.
Default is 5.

Required Entries for an Image Verification Station (client/server configuration)

udp_send?:respawn:udp_send > udp_send.IVS_DBNAME.out

-m defines this process as an Image Verification Station.

-t target where target is the defined alias of the client system that will run the external SIM, MonSIM, SMS, AMS, OCS, or IVS.

-w targetname where targetname is the name of the external SIM, MonSIM, or station as assigned in the Oracle® database.

-i ## sets the interval value. The server must acknowledge message receipt within this time; if not acknowledged, a timeout will occur, initiating a resend. 1-99 seconds are valid. Default is 2 seconds.

-e ## is the number of times a message will resend before a "device off-line" alarm is logged.
Default is 5.

Required Entries for OCSs (client/server configuration)

udp_send?:respawn:udp_send > udp_send.OCS_DBNAME.out

Provides interface support to the OCS. ?? represents any number, usually sequential, used to make the alias unique.

-a defines the process as an OCS process. udp_send -a may not be configured when aws_send is also present in the script.

-t target where target is the defined alias of the client system that will run the external SIM, MonSIM, SMS, AMS, OCS, or IVS.

-w targetname where targetname is the name of the external SIM, MonSIM, or station as assigned in the Oracle® database.

-i ## sets the interval value. The server must acknowledge message receipt within this time; if not acknowledged, a timeout will occur, initiating a resend. 1-99 seconds are valid. Default is 2 seconds.

-e ## is the number of times a message will resend before a "device off-line" alarm is logged.
Default is 5.

Required Entries for OCSs (X-Terminal configuration)

aws_send?:respawn:udp_send > dp_send.OCS_DBNAME.out

Provides interface support to the OCS. Mutually exclusive with `udp_send.??` represents any number, usually sequential, used to make the alias unique.

`-t` is a required entry if running X-terminal OCSs.

`-w OCSname` where the OCS name is the same as that entered into the DCS DEVICES-NETWORK STATIONS-OCSs form.

`-m map_queue_name` precedes the system name of the map_queue.

`-i ##` sets the message interval. The server must acknowledge message receipt within this time; if not acknowledged, a timeout will occur, initiating a resend. 1-99 seconds are valid. Default is 3.

`-e ##` is the number of times a message will resend before an alarm is logged. Default is 3.

However, each SIM active on the system must still have a corresponding `udp_send` process (without the `-a` parameter).

Example 1: Single Server , Client/Server Stations

```
ora_init:wait:ora_init > ora_init.out
alm_proc:respawn:alm_proc > alm_proc.out
alm_servr:respawn:alm_servr > alm_servr.out
amag_dbase:respawn:amag_dbase > amag_dbase.out
cctv_dummy:respawn:cctv_dummy > cctv_dummy.out
cctv_if:off:cctv_if -t lantronix -s ad -v ad > cctv_if.out
cmd_interp:respawn:cmd_interp > cmd_interp.out
cmd_proc:respawn:cmd_proc > cmd_proc.out
gtour_proc:respawn:gtour_proc > gtour_proc.out
icom_if:respawn:icom_if -dummy > icom_if.out
inc_update:respawn:inc_update -il > inc_update.out
log_servr:respawn:log_servr > log_servr.out
logtbl_proc:respawn:logtbl_proc -e 50000 -a 100000 > logtbl_proc.out
useorlose:off:useorlose 10 > useorlose.out
ora_intrf:respawn:ora_intrf > ora_intrf.out
ping_aws:respawn:ping_aws > ping_aws.out
recv_serv:respawn:recv_serv > recv_serv.out
aws_info:respawn:aws_info > aws_info.out
rts_lib:respawn:rts_lib > rts_lib.out
udp_recv:respawn:udp_recv > udp_recv.out
udp_send01:respawn:udp_send -a -t buffy -w HOCS1 > udp_send.HOCS1.out
udp_send02:respawn:udp_send -r -t buffy -w HRTS1 > udp_send.HRTS1.out
udp_send03:respawn:udp_send -a -t scotest2 -w XOCS1 >
udp_send.XOCS1.out
udp_send04:respawn:udp_send -a -t scotest -w XOCS2 > udp_send.XOCS2.out
udp_send05:respawn:udp_send -n -t buffy -w HSIM1 > udp_send.HSIM1.out
udp_send06:respawn:udp_send -n -t sim2 -w HYDSIM1 >
udp_send.HYDSIM1.out
udp_send07:respawn:udp_send -n -t sim2 -w XSIM1 > udp_send.XSIM1.out
nim_dload07:respawn:nim_dload -t sim2 -s XSIM1 > nim_dload.XSIM1.out
nim_dload05:respawn:nim_dload -t buffy -s HSIM1 > nim_dload.HSIM1.out
nim_dload06:respawn:nim_dload -t sim2 -s HYDSIM1 >
nim_dload.HYDSIM1.out
```

Example 2: Single Server, X-Term Stations

```
ora_init:wait:ora_init > ora_init.out
alm_proc:respawn:alm_proc > alm_proc.out
alm_servr:respawn:alm_servr > alm_servr.out
amag_dbase:respawn:amag_dbase > amag_dbase.out
cctv_dummy:respawn:cctv_dummy > cctv_dummy.out
```

```

cctv_if:off:cctv_if -t lantronix -s vicon -v vicon > cctv_if.out
cmd_interp:respawn:cmd_interp > cmd_interp.out
cmd_proc:respawn:cmd_proc > cmd_proc.out
gtour_proc:respawn:gtour_proc > gtour_proc.out
icom_if:respawn:icom_if -dummy > icom_if.out
inc_update:respawn:inc_update -il > inc_update.out
log_servr:respawn:log_servr -no_print > log_servr.out
logtbl_proc:respawn:logtbl_proc -e 50000 -a 100000 > logtbl_proc.out
useorlose:off:useorlose 10 > useorlose.out
ora_intrf:respawn:ora_intrf -d > ora_intrf.out
ping_aws:respawn:ping_aws > ping_aws.out
recv_serv:respawn:recv_serv > recv_serv.out
aws_info:respawn:aws_info > aws_info.out
rts_lib:respawn:rts_lib > rts_lib.out
udp_recv:respawn:udp_recv > udp_recv.out
aws_send:respawn:/e2000/ocs/aws_send -t -h buffy -w XOCS1 -m XOCS1_map
> aws_send.out
udp_send01:off:udp_send -a -t buffy -w HOCS1 > udp_send.HOCS1.out
udp_send02:respawn:udp_send -r -t buffy -w HRTS1 > udp_send.HRTS1.out
udp_send03:off:udp_send -v -t buffy -w HIVS1 > udp_send.HIVS1.out
udp_send04:off:udp_send -v -t test -w RIVS1 > udp_send.RIVS1.out
udp_send05:respawn:udp_send -m -t buffy -w HAMS1 > udp_send.HAMS1.out
udp_send06:off:udp_send -a -t scotest2 -w XOCS1 > udp_send.XOCS1.out
udp_send07:off:udp_send -a -t scosim2 -w XOCS2 > udp_send.XOCS2.out
udp_send08:off:udp_send -a -t scotest3 -w TOCS1 > udp_send.TOCS1.out
udp_send09:respawn:udp_send -n -t buffy -w HSIM1 > udp_send.HSIM1.out
udp_send10:off:udp_send -n -t sim2 -w HSIM1 > udp_send.HSIM1.out
udp_send11:off:udp_send -n -t sim2 -w XSIM1 > udp_send.XSIM1.out
udp_send12:off:udp_send -a -t ntserver -w NT > udp_send.NT.out
udp_send13:respawn:udp_send -n -t sim2 -w HYDSIM1 >
udp_send.HYDSIM1.out
nim_dload07:respawn:nim_dload -t buffy -s HSIM1 > nim_dload.HSIM1.out
nim_dload08:off:nim_dload -t sim2 -s HSIM1 > nim_dload.HSIM1.out
nim_dload09:off:nim_dload -t sim2 -s XSIM1 > nim_dload.XSIM1.out
nim_dload10:respawn:nim_dload -t sim2 -s HYDSIM1 >
nim_dload.HYDSIM1.out

```

Example 3: Redundant Server, X-Term Stations

```

ora_init:wait:ora_init > ora_init.out
alm_proc:respawn:alm_proc > alm_proc.out
alm_servr:respawn:alm_servr > alm_servr.out
amag_dbase:respawn:amag_dbase > amag_dbase.out
cctv_dummy:respawn:cctv_dummy > cctv_dummy.out
cctv_if:off:cctv_if -t lantronix -s vicon -v vicon > cctv_if.out
cmd_interp:respawn:cmd_interp > cmd_interp.out
cmd_proc:respawn:cmd_proc > cmd_proc.out
gtour_proc:respawn:gtour_proc > gtour_proc.out
icom_if:respawn:icom_if -dummy > icom_if.out
inc_update:respawn:inc_update -il > inc_update.out
log_servr:respawn:log_servr -no_print > log_servr.out
logtbl_proc:respawn:logtbl_proc -e 50000 -a 1000000 > logtbl_proc.out
useorlose:off:useorlose 10 > useorlose.out
ora_intrf:respawn:ora_intrf > ora_intrf.out
ping_aws:respawn:ping_aws > ping_aws.out
recv_serv:respawn:recv_serv > recv_serv.out
aws_info:respawn:aws_info > aws_info.out
rts_lib:respawn:rts_lib > rts_lib.out

```

```

udp_rcv:respawn:udp_rcv > udp_rcv.out
aws_send:off:/e2000/ocs/aws_send -t -h buffy -w XOCS1 -m XOCS1_map >
aws_send.out
udp_send01:respawn:udp_send -a -t buffy -w HOCS1 > udp_send.HOCS1.out
udp_send02:respawn:udp_send -r -t buffy -w HRTS1 > udp_send.HRTS1.out
udp_send03:respawn:udp_send -v -t buffy -w HIVS1 > udp_send.HIVS1.out
udp_send04:off:udp_send -v -t ivs -w RIVS1 > udp_send.RIVS1.out
udp_send05:respawn:udp_send -m -t buffy -w HAMS1 > udp_send.HAMS1.out
udp_send06:off:udp_send -a -t scotest2 -w XOCS1 > udp_send.XOCS1.out
udp_send07:off:udp_send -a -t scosim2 -w XOCS2 > udp_send.XOCS2.out
udp_send08:off:udp_send -a -t scotest3 -w TOCS1 > udp_send.TOCS1.out
udp_send09:respawn:udp_send -n -t buffy -w HSIM1 > udp_send.HSIM1.out
udp_send10:off:udp_send -n -t sim2 -w HSIM1 > udp_send.HSIM1.out
udp_send11:off:udp_send -n -t sim2 -w XSIM1 > udp_send.XSIM1.out
udp_send12:off:udp_send -a -t ntserver -w NT > udp_send.NT.out
udp_send13:respawn:udp_send -n -t sim2 -w HYDSIM1 >
udp_send.HYDSIM1.out
nim_dload07:respawn:nim_dload -t buffy -s HSIM1 > nim_dload.HSIM1.out
nim_dload08:off:nim_dload -t sim2 -s HSIM1 > nim_dload.HSIM1.out
nim_dload09:off:nim_dload -t sim2 -s XSIM1 > nim_dload.XSIM1.out
nim_dload10:respawn:nim_dload -t sim2 -s HYDSIM1 >
nim_dload.HYDSIM1.out

```

sim_mon

Script Entries

01:wait:init_nim > f.out

Performs the initial startup of the SIM and connects to the CCM.

- h servername identifies the name of the server the SIM will communicate with.
- s databasename identifies the name of the SIM as entered in the database.
- t externalSIMname identifies the UNIX alias of the machine on which the external SIM runs. This entry is required to identify external SIMs to the system, but is not needed for SIMs running on the system server.

02:respawn:amag_ctree > amag_ctree.out

Creates the SIM database structure.

03:respawn:dload_serv > dload_serv.out

Loads database records into the SIM database.

04:respawn:nim_rcv > nim_rcv.out

Receives commands generated from the server or OCS (i.e., door control, scheduled events) and passes them to the appropriate field controller.

05:respawn:nim_send > nim_send.out

Sends messages received by the amag_ctrl process to the server.

- h servername identifies the name of the system server.
- s databasename identifies the name of the SIM as entered in the system database.

10:respawn:amag_ctrl SITEX > ctrl.out

Valid for direct connect SIMs only. Establishes communications with and polls the field panels. SITEX is a defined communications loop in the system database (normally Site01, Site02, etc.) There will be one

amag_ctrl process defined for each loop monitored by the SIM.

-# where # is the total number of communications lines monitored by the master field controller.

-f configures the SIM for fast download.

08:off:amag_ctrl DIAL# > dial#.out

Set to Off for direct connect SIMs. For dialup SIMs, establishes communications with the field panels.

-# represents the number of the COM port on which communications takes place.

-f configures the SIM for fast download.

B is used to define a full duplex connection

09:off:master_ctrl DIAL# > master.out

Set to Off for direct connect SIMs. Determines and establishes communications with the master field panel.

-# represents the number of the COM port on which communications takes place.

Example Script 1

```
01:wait:init_nim -h moto -s SIM1 > init_nim.out
02:respawn:amag_ctree > amag_ctree.out
03:respawn:dload_serv > dload_serv.out
04:respawn:nim_recv > nim_recv.out
05:respawn:nim_send -h moto > nim_send.out
08:off:amag_ctrl DIAL# B -f > dial#.out
09:off:master_ctrl DIAL# > master.out
10:respawn:amag_ctrl SITE2 -f > site2.out
11:respawn:amag_ctrl SITE1 -f > ctrl.out
```

Example Script 2

```
01:wait:init_nim -h buffy -s HSIM1 > init_nim.out
02:respawn:amag_ctree > amag_ctree.out
03:respawn:dload_serv > dload_serv.out
04:respawn:nim_recv > nim_recv.out
05:respawn:nim_send -h buffy > nim_send.out
08:off:amag_ctrl DIAL3 B -f > dial3.out
09:off:master_ctrl DIAL3 > master.out
10:respawn:amag_ctrl QA_MULTI -f > ctrl4.out
11:respawn:amag_ctrl QA_MICRO -f > ctrl3.out
12:off:amag_ctrl QALOOP3 -f > ctrl1p3.out
13:off:amag_ctrl QALOOP4 -f > ctrl1p4.out
14:off:amag_ctrl SITE1 > ctrl1.out
```

ocs_mon

Script Entries

A separate file, named with the system name of the OCS (e.g., XOCS1_mon, OCS2_mon) where XOCS1 is the name of the OCS in the system) must be present in the /usr/ocs directory for each and every OCS supported by the system.

init_aws20:wait:init_aws20 > init_aws.out

Creates the files in /usr/amm/amm_data_files used by the CCM for processing.

-h servername identifies the name of the system server. This flag is not valid for redundant configurations.

-w OCSname identifies the OCS name as defined in the system database, for utilization in the team_start and team_stop processes.

-m mapqueueName is required if maps are to be displayed on alarm generation. The map queue name must match that found in either the aws_send or udp_send process entries in amm_mon.

aws_send:respawn:aws_send > aws_send.out

This entry is valid for client-server configurations only. It establishes the communication process sending OCS input to the CCM. It processes logins, alarm acknowledgements, and device control requests. NOTE: aws_send is used by both the OCS & the SMS. It should not be started more than once.

-h servername identifies the name of the system server. This flag is not valid for redundant configurations.

-w OCSname identifies the OCS name as defined in the system database, for utilization in the team_start and team_stop processes).

-m mapqueueName is required if maps are to be displayed on alarm generation. The map queue name must match that found in either the aws_send or udp_send process entries in amm_mon.

aws_recv:respawn:aws_recv > aws_recv.out

This entry is valid for client-server configurations only; receives alarms from the CCM.

-a defines the client as an OCS.

-w OCSname identifies the OCS name as defined in the system database, for utilization in the team_start and team_stop processes).

-i ## sets the logout interval of the OCS; resets to 0 when a message is received by the OCS. Default is 90 sec.

-m mapqueueName is required if maps are to be displayed on alarm generation. The map queue name must match that found in either the aws_send or udp_send process entries in amm_mon.

aws_mt:respawn:aws_mt > aws_mt.out

Displays the alarm scoreboard.

-s starts the OCS in silent mode (no beep when alarms appear on the OCS)

-w OCSname identifies the OCS name as defined in the system database, for utilization in the team_start and team_stop processes).

-i ## sets the logout interval of the OCS; resets to 0 when a message is received by the OCS. Default is 30 sec.

-m mapqueueName is required if maps are to be displayed on alarm generation. The map queue name must match that found in either the aws_send or udp_send process entries in amm_mon.

-t starts the OCS in xterm mode (requires aws_send instead of udp_send in amm_mon).

aws_maps:respawn:maps > maps.out

Creates the map window and requests for map display.

-h servername identifies the name of the server to which the OCS connects.

-w OCSname identifies the OCS name as defined in the system database, for utilization in the team_start and team_stop processes).

-q mapqueueName is required if maps are to be displayed on alarm generation. The map queue

name must match that found in either the aws_send or udp_send process entries in amm_mon.

Example Script 1: (X-Terminal OCSs)

```
init_aws20:wait:init_aws20 -m -h moto -w XOCS1 >init_aws.XOCS1.out  
aws_mt:respawn:aws_mt -w XOCS1 -m XOCS1_map -t > aws_mt.XOCS1.out  
aws_maps:respawn:maps -w XOCS1 -q XOCS1_map > maps.XOCS1.out
```

Example Script 2: (Client/Server OCSs)

```
init_aws20:wait:init_aws20 -m -h host1 -w OCS1 > init_aws.out  
aws_mt:respawn:aws_mt -t -w OCS1 -m OCS1_map -s > aws_mt.out  
aws_maps:respawn:maps -w OCS1 -q OCS1_map > maps.out  
init_aws21:wait:init_aws20 > init_aws.out
```

sms_mon

Script Entries

rts_init:wait:rts_init > rts_init.out

Starts the Status Monitoring Station processes.

-h servername identifies the name of the server to which the SMS connects.

-w stationname identifies the name of the SMS client.

sms_mt:respawn:rts_mt > rts_mt.out

Mounts the Status Monitor Station and processes user interactions.

-e places the process in debug mode.

-w stationname identifies the name of the SMS client.

rts_recv:respawn:rts_recv > rts_recv.out

Receives information from the CCM for transmission to the Status Monitor Station.

-r defines the process to start the Status Monitor Station.

-w stationname identifies the name of the SMS client.

rts_send:respawn:rts_send > rts_send.out

Sends information from the SMS to the CCM.

-h servername identifies the name of the server to which the SMS connects.

-m mapqueue name is required if maps are to be displayed on alarm generation. The map queue name must match that found in either the aws_send or udp_send process entries in amm_mon.

Example Script

```
rts_init:wait:rts_init -h buffy -w HRTS1 -d > rts_init.out  
rts_mt:respawn:rts_mt -w HRTS1 -e > rts_mt.out  
rts_recv:respawn:rts_recv -r -w HRTS1 -d > rts_recv.out  
rts_send:respawn:rts_send -h buffy -m HRTS1_map > rts_send.out
```

ivs_mon

Script Entries

ivs20:wait:ivs_init > ivs_init.out

Starts the Image Verification Station processes.

-h servername identifies the name of the server to which the IVS connects.

-w stationname identifies the name of the IVS client.

ivs_send:respawn:ivs_send > ivs_send.out

Mounts the Image Verification Station and processes user interactions.

-e places the process in debug mode.

-h servername identifies the name of the IVS server.

ivs_rcv:respawn:ivs_rcv > ivs_rcv.out

Receives information from the CCM for transmission to the Activity Monitor Station.

-w stationname identifies the name of the SMS client

ivs_mt:respawn:ivs_mt > ivs_mt.out

Sends information from the SMS to the CCM.

-w clientname is the name of the IVS client system.

Example Script

```
ivs20:wait:ivs_init -h buffy -w HIVS1 > ivs_init.out
ivs_send:respawn:ivs_send -h buffy > ivs_send.out
ivs_rcv:respawn:ivs_rcv -w HIVS1 > ivs_rcv.out
ivs_mt:respawn:ivs_mt -w HIVS1 -d > ivs_mt.out
```

ams_mon

Script Entries

ams20:wait:init_ams > init_ams.out

Starts the Activity Monitoring Station processes.

-h servername identifies the name of the server to which the AMS connects.

-w stationname identifies the name of the AMS client.

ams_send:respawn:ams_send > ams_send.out

Mounts the Activity Monitor Station and processes user interactions.

-e places the process in debug mode.

-w stationname identifies the name of the AMS client.

ams_rcv:respawn:ams_rcv > ams_rcv.out

Receives information from the CCM for transmission to the Activity Monitor Station.

-r defines the process to start the Activity Monitor Station

-w stationname identifies the name of the SMS client

ams_send:respawn:ams_send > ams_send.out

Sends information from the SMS to the CCM.

-h servername identifies the name of the server to which the SMS connects

-m mapqueueName is required if maps are to be displayed on alarm generation. The map queue name must match that found in either the aws_send or udp_send process entries in

amm_mon.

Example Script

```
ams20:wait:init_ams -d -w DILBERT_AMS -h dilbert > ams.init.out  
ams_send:respawn:ams_send -d -h dilbert > ams_send.out  
ams_recv:respawn:ams_recv -d -w DILBERT_AMS > ams_recv.out  
ams_mt:respawn:ams_mt -w DILBERT_AMS -d > ams_mt.out
```