Developing Agents With the Generic Data Service (GDS)

Part No: E69328
August 2018

ORACLE

Developing Agents With the Generic Data Service (GDS)

Part No: E69328

Référence: E69328

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Describes how to install and configure the Oracle Solaris Cluster Generic Data Service (GDS) to create a highly available, custom Oracle Solaris Cluster data service.
- **Audience** – Technicians, system administrators, and authorized service providers
- **Required knowledge** – Advanced experience troubleshooting and replacing hardware

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E69294_01.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

Creating a Data Service with GDS

This book provides information about generic data services (GDS), and shows you how to create a data service that uses the GDS. You create this service by using either Oracle Solaris Cluster Agent Builder or Oracle Solaris Cluster administration commands.

The following two versions of GDS are supported:

- Version 1 of the GDS – GDS
- Version 2 of the GDS – GDSv2

This chapter covers the following topics:

Generic Data Service Concepts

The GDS is a mechanism for making simple network-aware and non-network-aware applications highly available or scalable by plugging them into the Oracle Solaris Cluster Resource Group Management (RGM) framework. This mechanism does not require you to code a data service, which you typically must do to make an application highly available or scalable.

Note - You can install and configure this data service to run in either the global zone or a zone cluster. For updated information about supported configurations of this data service, see the *Oracle Solaris Cluster 4 Compatibility Guide*.

The GDS is a single, precompiled data service. You cannot modify the precompiled data service and its components, the callback method (`rt_callbacks`) implementations, and the resource type registration file (`rt_reg`).

This section covers the following topics:

Precompiled Resource Type

The generic data service resource type `SUNW.gds` is included in the `ha-cluster/ha-service/gds` package. The `ha-cluster/ha-service/gds` package includes the following files:

```
# pkg contents ha-cluster/ha-service/gds

PATH
/opt/SUNWscgds
/opt/SUNWscgds/bin
/opt/SUNWscgds/bin/gds_monitor_check
/opt/SUNWscgds/bin/gds_monitor_start
/opt/SUNWscgds/bin/gds_monitor_stop
/opt/SUNWscgds/bin/gds_probe
/opt/SUNWscgds/bin/gds_svc_start
/opt/SUNWscgds/bin/gds_svc_stop
/opt/SUNWscgds/bin/gds_update
/opt/SUNWscgds/bin/gds_validate
/opt/SUNWscgds/etc
/opt/SUNWscgds/etc/SUNW.gds
/opt/cluster
/opt/cluster/lib
/opt/cluster/lib/rgm
/opt/cluster/lib/rgm/rtreg
/opt/cluster/lib/rgm/rtreg/SUNW.gds
```

Advantages and Disadvantages of Using the GDS

Using the GDS has the following advantages over using either the Agent Builder source code (see the `scdscreate`(8HA) man page) or Oracle Solaris Cluster administration commands:

- The GDS is easy to use.
- The GDS and its methods are precompiled and therefore cannot be modified.
- You can use Agent Builder to generate scripts for your application. These scripts are put in an Oracle Solaris package that can be reused across multiple clusters.

While using the GDS has many advantages, the GDS is *not* the mechanism to use in these instances:

- When more control is required than is available with the precompiled resource type, such as when you need to add extension properties or change default values
- When the source code needs to be modified to add special functions

Ways to Create a Service That Uses the GDS

There are two ways to create a service that uses the GDS:

- Agent Builder
- Oracle Solaris Cluster administration commands

GDS and Agent Builder

Use Agent Builder and select GDS as the type of generated source code. The user input is used to generate a set of scripts that configure resources for the given application. For more information, see Chapter 3, “Using Agent Builder to Create a Service That Uses GDS or GDSv2”.

GDS and Oracle Solaris Cluster Administration Commands

This method uses the precompiled data service code in `ha-cluster/ha-service/gds`. However, the cluster administrator must use Oracle Solaris Cluster administration commands to create and configure the resource. See the `clresource`(8CL) man page.

Selecting the Method to Use to Create a GDS-Based Service

A significant amount of typing is required to issue Oracle Solaris Cluster commands. For example, see “How to Use Oracle Solaris Cluster Administration Commands to Create a Highly Available Service That Uses the GDS” on page 18 and “How to Use Oracle Solaris Cluster Administration Commands to Create a Scalable Service That Uses the GDS” on page 19.

Using the GDS with Agent Builder simplifies the process because the GDS generates the scripts that issue the `scrgadm` and `scswitch` commands for you.

How the GDS Logs Events

The GDS enables you to log relevant information that is passed from the GDS to the scripts that the GDS starts. This information includes the status of the start, probe, validate, and stop methods as well as property variables. You can use this information to diagnose problems or errors in your scripts, or apply it to other purposes.

You use the `Log_level` property that is described in "`Log_level` Property" on page 14 to specify the level, or type, of messages that the GDS will log. You can specify `NONE`, `INFO`, or `ERR`.

GDS Log Files

The following two GDS log files are placed in the `/var/cluster/logs/DS/`*resource-group-name*`/`*resource-name* directory:

- The `start_stop_log.txt`, which contains messages that are generated by resource start and stop methods
- The `probe_log.txt`, which contains messages that are generated by the resource monitor

The following example shows the types of information that `start_stop_log.txt` contains:

```
06/12/2006 12:38:05 phys-node-1 START-INFO> Start succeeded. [/home/brianx/sc/start_cmd]
06/12/2006 12:42:11 phys-node-1 STOP-INFO> Successfully stopped the application
```

The following example shows the types of information that `probe_log.txt` contains:

```
06/12/2006 12:38:15 phys-node-1 PROBE-INFO> The GDS monitor (gds_probe) has been started
06/12/2006 12:39:15 phys-node-1 PROBE-INFO> The probe result is 0
06/12/2006 12:40:15 phys-node-1 PROBE-INFO> The probe result is 0
06/12/2006 12:41:15 phys-node-1 PROBE-INFO> The probe result is 0
```

Required GDS Properties

This section describes the required GDS properties.

`Port_list` Property

The `Port_list` property identifies the list of ports on which the application listens. You must specify the `Port_list` property in the start script that Agent Builder creates or with the `clresource` command.

Whether you must specify this property depends on whether your application is network aware or not. If you specify that your application is network aware (you set the `Network_aware` property to `TRUE`, which is the default), you must provide both the `Start_command` extension property and the `Port_list` property. If you specify that your application is non-network aware (you set the `Network_aware` property to `FALSE`), you must provide only the `Start_command` extension property. The `Port_list` property is optional.

`Start_command` Property

The start command, which you specify with the `Start_command` extension property, starts the application. This command must be a UNIX command with arguments that can be passed directly to a shell to start the application.

If your application is network aware, you must provide both the `Start_command` extension property and the `Port_list` property. If your application is non-network aware, you must provide only the `Start_command` extension property.

Optional GDS Properties

Optional GDS properties include both *standard properties* and *extension properties*. Standard properties are a standard set of properties that are provided by Oracle Solaris Cluster. Properties that are defined in the RTR file are called extension properties.

Optional GDS properties include:

- `Child_mon_level` extension property (used only with administration commands)
- `Failover_enabled` extension property
- `Log_level` extension property
- `Monitor_retry_count` extension property
- `Monitor_retry_interval` extension property
- `Network_aware` extension property
- `Probe_command` extension property
- `Probe_timeout` extension property
- `Resource_dependencies` property
- `Start_timeout` property
- `Stop_command` extension property
- `Stop_signal` extension property
- `Stop_timeout` property
- `Timeout_threshold` property

- `Validate_command` extension property
- `Validate_timeout` property

`Child_mon_level` Property

Note - If you use Oracle Solaris Cluster administration commands, you can use the `Child_mon_level` property. If you use Agent Builder, you cannot use this property.

This property provides control over the processes that are monitored through the Process Monitor Facility (PMF). This property denotes the level up to which the forked children processes are monitored. This property works like the `-C` argument to the `pmfadm` command. See the `pmfadm`(8) man page.

Omitting this property, or setting it to the default value of `-1`, has the same effect as omitting the `-C` option on the `pmfadm` command. That is, all children and their descendants are monitored.

`Failover_enabled` Property

This property controls the failover behavior of the resource. If this extension property is set to `TRUE`, the application fails over when the number of restarts exceeds the `Retry_count` within the `Retry_interval` number of seconds.

If this property is set to `FALSE`, the application does not restart or fail over to another node when the number of restarts exceeds the `Retry_count` within the `Retry_interval` number of seconds.

You can use this property to prevent the application resource from initiating a failover of the resource group. The default value for this property is `TRUE`.

Note - In future, use the `Failover_mode` property in place of the `Failover_enabled` extension property as `Failover_mode` better controls failover behavior. For more information, see the descriptions of the `LOG_ONLY` and `RESTART_ONLY` values for `Failover_mode` in the `r_properties`(7) man page.

`Log_level` Property

This property specifies the level, or type, of diagnostic messages that are logged by the GDS. You can specify `NONE`, `INFO`, or `ERR` for this property. When you specify `NONE`, diagnostic messages are not logged by the GDS. When you specify `INFO`, only informational messages are

logged. When you specify `ERR`, only error messages are logged. By default, the GDS does not log diagnostic messages (`NONE`).

`Monitor_retry_count` Property

This property specifies the number of times that the process monitor facility (PMF) restarts the fault monitor during the time window that the `Monitor_retry_interval` property specifies. This property refers to restarts of the fault monitor itself rather than to the resource. The system-defined properties `Retry_interval` and `Retry_count` control restarting of the resource.

`Monitor_retry_interval` Property

This property specifies the time (in minutes) over which failures of the fault monitor are counted. If the number of times that the fault monitor fails exceeds the value that is specified in the extension property `Monitor_retry_count` within this period, the PMF does not restart the fault monitor.

`Network_aware` Property

This property specifies whether your application uses the network. By default, the GDS assumes that your application is network aware, that is, uses the network (`Network_aware` is set to `TRUE`).

If your application is network aware, you must provide both the `Start_command` extension property and the `Port_list` property. If your application is non-network aware, you must provide only the `Start_command` extension property.

`Probe_command` Property

This property specifies the probe command that periodically checks the health of a given application. This command must be a UNIX command with arguments that can be passed directly to a shell to probe the application. The probe command returns with an exit status of `0` if the application is running correctly.

The exit status of the probe command is used to determine the severity of the application's failure. This exit status, called the *probe status*, must be an integer between `0` (for success) and `100` (for complete failure). The probe status can also be a special value of `201`, which causes the application to immediately fail over unless `Failover_enabled` is set to `FALSE`. The GDS

probing algorithm uses the probe status to determine whether to restart the application locally or fail it over. See the `scds_fm_action`(3HA) man page for more information. If the exit status is `201`, the application is immediately failed over.

If the probe command is omitted, the GDS provides its own simple probe. This probe connects to the application on the set of IP addresses that is derived from the output of the `scds_get_netaddr_list()` function. This includes all network resources on which the GDS resource declares a resource dependency. If there are no such resources, it includes all network resources configured in the same resource group as the GDS resource. See the `scds_get_netaddr_list`(3HA) man page for more information.

If the connect succeeds, the connect disconnects immediately. If both the connect and disconnect succeed, the application is deemed to be running well.

Note - The probe that is provided with the GDS is only intended to be a simple substitute for the fully functioning application-specific probe.

`Probe_timeout` Property

This property specifies the timeout value for the probe command. See “`Probe_command` Property” on page 15 for additional information. The default for `Probe_timeout` is 30 seconds.

`Resource_dependencies` Property

This property specifies a list of resources in the same group or in different groups upon which this resource has a strong dependency. This resource cannot be started if the start of any resource in the list fails. If this resource and one of the resources in the list start at the same time, the RGM waits until the resource in the list starts before the RGM starts this resource. If the resource in this resource's `Resource_dependencies` list does not start (for example, if the resource group for the resource in the list remains offline or if the resource in the list is in a `Start_failed` state), this resource also remains offline. If this resource remains offline because of a dependency on a resource in a different resource group that fails to start, this resource's group enters a `Pending_online_blocked` state.

To specify the scope of a dependency, append the qualifier `{ANY_NODE}`, `{FROM_RG_AFFINITIES}`, `{LOCAL_NODE}`, or `@`*nodename*, including the braces ({}) or at-sign (@), to the resource name when you specify this property.

See `Resource_dependencies` in the `r_properties`(7) man page for details about resource dependencies.

`Start_timeout` Property

This property specifies the start timeout for the start command. See “`Start_command` Property” on page 13 for additional information. The default for `Start_timeout` is 300 seconds.

`Stop_command` Property

This property specifies the command that must stop an application and only return after the application has been completely stopped. This command must be a complete UNIX command that can be passed directly to a shell to stop the application.

If the `Stop_command` extension property is provided, the GDS stop method starts the stop command with 80 percent of the stop timeout. Regardless of the outcome of starting the stop command, the GDS stop method sends `SIGKILL` with 15 percent of the stop timeout. The remaining 5 percent of the time is reserved for housekeeping overhead.

If the stop command is omitted, the GDS tries to stop the application by using the signal specified in `Stop_signal`.

`Stop_signal` Property

This property specifies a value that identifies the signal to stop an application through the PMF. See the `signal`(3HEAD) man page for a list of the integer values that you can specify. The default value is `15` (`SIGTERM`).

`Stop_timeout` Property

This property specifies the timeout for the stop command. See “`Stop_command` Property” on page 17 for additional information. The default for `Stop_timeout` is 300 seconds.

`Timeout_threshold` Property

This property specifies after what percentage of a timeout period a notification should be sent that the timeout limit is almost reached.

`Validate_command` Property

This property specifies the absolute path to a command to invoke to validate the application. If you do not provide an absolute path, the application is not validated.

`Validate_timeout` Property

This property specifies the timeout for the validate command. See "`Validate_command` Property" on page 18 for additional information. The default for `Validate_timeout` is 300 seconds.

Using Oracle Solaris Cluster Administration Commands to Create a Service That Uses the GDS

This section describes how to input arguments to the GDS. You use the existing Oracle Solaris Cluster administration commands, such as `clresourcetype`, `clresourcegroup`, and `clresource` to maintain and administer the GDS.

If the scripts provide adequate functionality, you do not need to use the lower-level administration commands that are shown in this section. However, you can use the lower-level administration commands if you need to have finer control over the GDS-based resource. These commands are executed by the scripts.

▼ How to Use Oracle Solaris Cluster Administration Commands to Create a Highly Available Service That Uses the GDS

Before You Begin Ensure that the `/etc/netmasks` file has IP-address subnet and netmask entries for all logical hostnames. If necessary, edit the `/etc/netmasks` file to add any missing entries.

1. **Become an administrator that provides `solaris.cluster.modify` authorization.**

2. **Register the resource type `SUNW.gds`.**

```
# clresourcetype register SUNW.gds
```

3. **Create the resource group that contains the `LogicalHostname` resource and the failover service itself.**

```
# clresourcegroup create haapp_rg
```

4. **Create the resource for the `LogicalHostname` resource.**

```
# clreslogicalhostname create -g haapp_rg hhead
```

5. **Create the resource for the failover service itself.**

```
# clresource create -g haapp_rg -t SUNW.gds
-p Validate_command="/export/app/bin/configtest" \
-p Scalable=false \
-p Start_timeout=120 \
-p Stop_timeout=120 \
-p Probe_timeout=120 \
-p Port_list="2222/tcp" \
-p Start_command="/export/ha/appctl/start" \
-p Stop_command="/export/ha/appctl/stop" \
-p Probe_command="/export/app/bin/probe" \
-p Child_mon_level=0 \
-p Resource_dependencies=hhead \
-p Failover_enabled=TRUE \
-p Stop_signal=9 haapp_rs
```

Note - The scripts listed above are examples; your script names might be different.

6. **Bring the resource group `haapp_rg` online in a managed state.**

```
# clresourcegroup online -M haapp_rg
```

How to Use Oracle Solaris Cluster Administration Commands to Create a Scalable Service That Uses the GDS

Before You Begin Ensure that the `/etc/netmasks` file has IP-address subnet and netmask entries for all logical hostnames. If necessary, edit the `/etc/netmasks` file to add any missing entries.

1. **Become an administrator that provides `solaris.cluster.modify` authorization.**

2. **Register the resource type `SUNW.gds`.**

```
# clresourcetype register SUNW.gds
```

3. **Create the resource group for the `SharedAddress` resource.**

```
# clresourcegroup create sa_rg
```

4. **Create the `SharedAddress` resource `hhead` in resource group `sa_rg`.**

```
# clressharedaddress create -g sa_rg hhead
```

5. **Create the resource group for the scalable service.**

```
# clresourcegroup create -S -p RG_dependencies=sa_reg app_rg
```

6. **Create the resource for the scalable service.**

```
# clresource create -g app_rg -t SUNW.gds
-p Validate_command="/export/app/bin/configtest" \
-p Scalable=TRUE -p Start_timeout=120 \
-p Stop_timeout=120 -p Probe_timeout=120 \
-p Port_list="2222/tcp" \
-p Start_command="/export/app/bin/start" \
-p Stop_command="/export/app/bin/stop" \
-p Probe_command="/export/app/bin/probe" \
-p Child_mon_level=0 -p Network_resource_used=hhead \
-p Failover_enabled=TRUE -p Stop_signal=9 app_rs
```

7. **Bring the resource group that contains the network resources online.**

```
# clresourcegroup online sa_reg
```

8. **Bring the resource group `app_rg` online in a managed state.**

```
# clresourcegroup online -M app_reg
```

♦ ♦ ♦ CHAPTER 2

Creating a Data Service with GDSv2

This chapter explains how to install and configure the GDSv2 and create a demo resource.

This chapter contains the following sections.

Overview of the GDSv2

Oracle Solaris Cluster supports both versions of GDS (GDS and GDSv2).

This section contains information about the following:

- The `ORCL.gds` and `ORCL.gds_proxy` resource types
- RGM callback methods for the GDSv2 resource types
- The *method*_`command` Sequence

Resource Types

The GDSv2 uses `ORCL.gds` and `ORCL.gds_proxy` resource types.

A proxy resource type is typically used to reflect the state of a resource from another cluster framework. The proxy resource type was initially developed to proxy state information of the Oracle RAC database running under the control of the Oracle Solaris Cluster Ready Service,

now known as Oracle Clusterware. However, a proxy resource type is not limited to proxying state information from another cluster framework and instead could reflect the state of any application. In the examples that are provided, the demo resource of type `ORCL.gds_proxy` reflects the state of the SMF system log service.

RGM Callback Methods

The `ORCL.gds` and `ORCL.gds_proxy` resource types include RGM callback methods and associated *method*`_command` extension properties.

The `ORCL.gds` resource type includes the following RGM callback methods and associated *method*`_command` extension properties:

RGM Callback Method	**GDSv2** *method*`_command`
Boot	`Boot_command`
Fini	`Fini_command`
Init	`Init_command`
Start	`Start_command`
Stop	`Stop_command`
Validate	`Validate_command`
Method_start	`Probe_command`
Method_stop	
Method_check	
Update	

The `ORCL.gds_proxy` resource type includes the following RGM callback methods and associated *method*`_command` extension properties:

RGM Callback Method	**GDSv2** *method*`_command`
Boot	`Boot_command`
Init	`Init_command`
Fini	`Fini_command`
Prenet_start	`Prenet_start_command`
Postnet_stop	`Postnet_stop_command`
Validate	`Validate_command`

The GDSv2 also includes the following:

- Useful housekeeping KSH function scripts for GDSv2 resource types.
- Demo resources of type `ORCL.gds` and `ORCL.gds_proxy` to showcase functionality.
- Enhanced GDSv2 probing algorithm to minimize probe timeouts.
- Enhanced Oracle Solaris Cluster Agent Builder GUI and CLI commands to create new resources of type `ORCL.gds` and `ORCL.gds_proxy`, as well as new resources from sub-classed `ORCL.gds` or `ORCL.gds_proxy` resource types.

The *method*_`command` Sequence

To see a complete list of all callback methods executed by the RGM, see "RGM Callback Methods" on page 22. The RGM callback method and subsequent GDSv2 *method*_`command` sequences are listed in the following sections.

The `ORCL.gds` *method*_`command` Sequence

The table below lists the `ORCL.gds` *method*_`command` extension properties.

Action	**RGM Callback Method**	**ORCL.gds** *method*_**command**
Resource creation	Validate	`Validate_command` If set, the `Validate_command` is executed on all nodes within the resource group's node list.
	Init	`Init_command` If set, the `Init_command` is executed on all nodes identified by the `Init_nodes` property.
Resource enable	Start	`Start_command` `Start_command` is a required property.
	Monitor start	`Probe_command` Monitor_start will only execute `Probe_command` if it is set. If `Probe_command` is not set but `PMF_managed=TRUE` is set, then Monitor_start will start an internal probe.
Resource disable	Monitor stop	`Probe_command` Monitor_stop will only stop the probe if `Probe_command` was set. If `Probe_command` was not set but `PMF_managed=TRUE` was set, then Monitor_stop will stop the internal probe.

Action	RGM Callback Method	ORCL.gds *method*_command
	Stop	`Stop_command` `Stop_command` is only executed if it is set. If `Stop_command` is not set but `PMF_managed=TRUE` is set, then `Stop_signal` is sent to the application process tree if `Stop_command` failed to stop the application.
Resource delete	Fini	`Fini_command` If set, `Fini_command` is executed on all nodes within the resource group's node list.
Resource unmonitor	Monitor stop	`Probe_command` Monitor_stop will only stop the probe if `Probe_command` was set. If `Probe_command` was not set but `PMF_managed=TRUE` was set, then Monitor_stop will stop the internal probe.
Resource monitor	Monitor start	`Probe_command` Monitor_start will only execute `Probe_command` if it is set. If `Probe_command` is not set but `PMF_managed=TRUE` is set, then Monitor_start will start an internal probe.
Property update for an enabled resource	Validate	`Validate_command` If set, the `Validate_command` is executed on all nodes within the resource group's node list.
	Update	On the node where the resource is online, the RGM `Update` method will kill the fault monitor process tree and then use PMF to restart the fault monitor.
Property update for a disabled resource	Validate	`Validate_command` If set, the `Validate_command` is executed on all nodes within the resource group's node list.
Upon reboot for an enabled and monitored resource	Boot	`Boot_command` If set, `Boot_command` is executed on all nodes identified by the `Init_nodes` property.
	Start	`Start_command` `Start_command` is a required property.
	Monitor Start	`Probe_command` Monitor_start will only execute `Probe_command` if it is set. If `Probe_command` is not set but `PMF_managed=TRUE` is set, then Monitor_start will start an internal probe.
Upon reboot for a disabled resource	Boot	`Boot_command` If set, `Boot_command` is executed on all nodes identified by the `Init_nodes` property.

The `ORCL.gds_proxy` *method*`_command` Sequence

The table below lists the `ORCL.gds_proxy` *method*`_command` extension properties.

Action	RGM Callback Method	ORCL.gds_proxy *method*_command
Resource creation	Validate	`Validate_command` If set, the `Validate_command` is executed on all nodes within the resource group's node list.
	Init	`Init_command` If set, the `Init_command` is executed on all nodes identified by the `Init_nodes` property.
Resource enable	Prenet start	`Prenet_start_command` `Prenet_start_command` is a required property.
Resource disable	Postnet stop	`Postnet_stop_command` If set, `Postnet_stop_command` is executed on the node where the resource is being disabled. If `Postnet_stop_command` is not set, then the `Stop_signal` property is sent to the proxy PMF tag.
Resource delete	Fini	`Fini_command` If set, `Fini_command` is executed on all nodes within the resource group's node list.
Upon reboot for an enabled resource	Boot	`Boot_command` If set, `Boot_command` is executed on all nodes identified by the `Init_nodes` property.
	Prenet start	`Prenet_start_command` `Prenet_start_command` is a required property.
Upon reboot for a disabled resource	Boot	`Boot_command` If set, `Boot_command` is executed on all nodes identified by the `Init_nodes` property.

The Resource Group *method*`_command` Sequence

The table below lists the resource group *method*`_command` extension properties.

Action	RGM Callback Method	RGM Callback Method and *method*_command
Resource group offline with an enabled resource	Postnet stop	`Postnet_stop_command` If set, `Postnet_stop_command` is executed on the node where the resource is being disabled. If `Postnet_stop_command` is not set, then the `Stop_signal` property is sent to the proxy PMF tag.
Resource group online with previously enabled resource	Prenet start	`Prenet_start_command` `Prenet_start_command` is a required property.
Resource group switch from unmanaged to managed	Init	`Init_command` If set, the `Init_command` is executed on all nodes identified by the `Init_nodes` property.
Resource group switch from managed to unmanaged	Fini	`Fini_command` If set, `Fini_command` is executed on all nodes within the resource group's node list.

Installing and Configuring the GDSv2

TABLE 1 Tasks for Installing and Configuring the GDSv2

Task	Instructions
Install the GDSv2 software	“Installing the GDSv2” on page 26
Configure the GDSv2 software	“Configuring the GDSv2” on page 27
Register the GDSv2 software	“Registering a GDSv2 Resource Type” on page 27
Create a GDSv2 Resource	“Creating a GDSv2 Resource” on page 28

Installing the GDSv2

This section contains information about how the GDSv2 software is installed.

The GDSv2 is automatically installed when you install any of the following Oracle Solaris Cluster packages:

- The ha-cluster-full package
- The ha-cluster-framework-full package

- The ha-cluster-data-services-full package

If you installed the ha-cluster-minimal group package, you can manually use the `pkg(1)` command to install the Oracle Solaris IPS package:

```
pkg://ha-cluster/ha-service/gds2
```

Configuring the GDSv2

This section contains information about configuring the GDSv2, which is performed by setting extension properties. In most cases, you can use the default values for the extension properties. For specific information on extension properties, see "Using the GDSv2 Extension Properties" on page 32.

Registering a GDSv2 Resource Type

This section contains the procedure to register a GDSv2 resource type.

▼ How to Register a GDSv2 Resource Type

1. **On one cluster node, assume the `root` role.**

2. **Register either the `ORCL.gds` or the `ORCL.gds_proxy` resource type.**

```
# clresourcetype register ORCL.gds

# clresourcetype register ORCL.gds_proxy
```

3. **Ensure that the resource type was registered.**

```
# clresourcetype list ORCL.gds

ORCL.gds:1

# clresourcetype list ORCL.gds_proxy

ORCL.gds_proxy:1
```

Creating a GDSv2 Resource

This section contains procedures to create a demo GDSv2 resource. A demo resource is used as a starting point for your own GDSv2 resource. The demo scripts are located in the GDSv2 package.

Note - The purpose of using demo applications is to showcase the behavior of the GDSv2 resource type. As such, these demo applications are just simple commands that are already installed and configured on Oracle Solaris 11.

The benefit of a demo application is to quickly deploy a GDSv2 resource with minimal effort. You can then experiment with the various GDSv2 extension properties to learn about the functionality.

The application used by the demo resource of type `ORCL.gds` executes a background sleep for 1800 seconds. After you implement that application, you make other customizations to the `ORCL.gds` resource type. The application used by the demo resource of type `ORCL.gds_proxy` reflects the status of the Solaris Service Management Facility (SMF) system-log.

▼ How to Create a Demo Resource of Type `ORCL.gds`

This procedure assumes you have already registered the `ORCL.gds` resource type. See "How to Register a GDSv2 Resource Type" on page 27.

1. **On one cluster node, assume the `root` role.**

2. **Create a failover resource group and a demo resource of type `ORCL.gds`.**

Note - A resource of type `ORCL.gds` requires that the `Start_command` extension property is used. All other extension properties are optional.

```
# clresourcegroup create -p pathprefix=/opt/ORCLscgds/demo myrg
# clresource create -g myrg -t ORCL.gds \
-p start_command="%RG_PATHPREFIX/demo_start \
-R %RS_NAME -G %RG_NAME -T %RT_NAME" -d myrs
```

These steps use the following optional property variables:

- %RG_PATHPREFIX – Determines the path for the `demo_start` script.
- %RS_NAME – Determines the resource name.

- %RG_NAME – Determines the resource group name.
- %RT_NAME – Determines the resource type name.

  GDSv2 replaces the *%Property_Variables* with the actual resource name, resource group name, and resource type name when executing the `demo_start` script. These global variables can *then* be used by the `scha_cmds(8HA)` commands. For example, within the `/opt/ORCLscgds/demo/demo_start` script, the following is used:

  ```
  /usr/cluster/bin/scha_resource_get -O extension -R ${RESOURCE} -G ${RESOURCE_GROUP}
  interpose_logical_hostname
  ```

3. Bring the resource online.

```
# clresourcegroup online -eM myrg
# clresource status myrs

=== Cluster Resources ===
Resource Name       Node Name      State        Status Message
-------------       ---------      -----        --------------
myrs                node1          Online       Online - Service is online
                    node2          Offline      Offline
```

4. Verify that the Oracle Solaris Cluster PMF is running and display the PMF tag information for the *myrs* resource.

By default, the `ORCL.gds` resource type uses Oracle Solaris Cluster's PMF. If the process that is being monitored fails, PMF immediately restarts the process. In the example below, process 3006 is the process that was started by the `demo_start` script. This process represents the demo application, `sleep 1800 &`. Two PMF tags are shown below: `myrg,myrs,0.mon` and `myrg, myrs,0.svc`.

```
# pmfadm -l ""
STATUS  myrg,myrs,0.mon
pmfadm -c myrg, myrs,0.mon -n 4 -t 2 /bin/ksh -c \
'/opt/ORCLscgds/bin/gds_probe -R myrs -T ORCL.gds -G myrg'
    retries: 0
    owner: root
    monitor children: all
    pids: 3020
STATUS  myrg,myrs,0.svc

pmfadm -c myrg, myrs,0.svc -a /usr/cluster/lib/sc/scds_pmf_action_script /bin/ksh -c \
'/usr/cluster/bin/hatimerun -t 299 /opt/ORCLscgds/demo/demo_start -R myrs -G myrg \
-T ORCL.gds ; echo $? > /var/cluster/run/tempubaG0f'
    retries: 0
    owner: root
    monitor children: all
```

```
pids: 3006
```

The PMF tag `myrg.myrs,0.mon` represents the GDSv2 monitor, and `myrg,myrs,0.svc`represents the GDSv2 application process. The PMF tag `myrg,myrs,0.svc` will disappear if all the application processes that are being monitored have failed. Consequently, if process 3006 dies (which it will eventually as process 3006 is `sleep 1800 &`), then the PMF immediately restarts the application. As a test, you can kill your equivalent process ID 3006 and reissue the `clresource status myrs` and `pmfadm -l ""` commands to see that the application was immediately restarted.

5. **Set additional *method*`_command` extension properties.**

   A resource of type `ORCL.gds` requires that you use the `start_command` extension property. This demo example uses additional *method*`_command` properties. You can also set these extension properties after the resource has been created. The steps below show how to set the properties during resource creation.

   a. **Disable and delete the resource.**

   ```
   # clresource disable myrs
   # clresource delete myrs
   ```

   b. **Create the resource.**

   ```
   # clresource create -g myrg -t ORCL.gds \
   -p Start_command="%RG_PATHPREFIX/demo_start -R %RS_NAME -G %RG_NAME -T %RT_NAME" \
   -p Stop_command="%RG_PATHPREFIX/demo_stop -R %RS_NAME -G %RG_NAME -T %RT_NAME" \
   -p Probe_command="%RG_PATHPREFIX/demo_probe -R %RS_NAME -G %RG_NAME -T %RT_NAME" \
   -p Validate_command="%RG_PATHPREFIX/demo_validate -R %RS_NAME -G %RG_NAME \
   -T %RT_NAME" -d myrs
   ```

   The demo resource of type `ORCL.gds` has been created.

6. **Enable the resource.**

   ```
   # clresource enable myrs
   ```

How to Create a Demo Resource of Type `ORCL.gds_proxy`

This procedure assumes you have already registered the `ORCL.gds_proxy` resource type. See "How to Register a GDSv2 Resource Type" on page 27.

1. **On one cluster node, assume the `root` role.**

2. **Create a scalable resource group and a demo resource of type `ORCL.gds_proxy`.**

```
# clresourcegroup create -p pathprefix=/opt/ORCLscgds/demo -S mysrg

# clresource create -g mysrg -t ORCL.gds_proxy \
-p Prenet_start_command="%RG_PATHPREFIX/demo_proxy_prenet_start \
-R %RS_NAME -G %RG_NAME -T %RT_NAME" \
-p Postnet_stop_command="%RG_PATHPREFIX/demo_proxy_postnet_stop \
-R %RS_NAME -G %RG_NAME -T %RT_NAME" \
-p Validate_command="%RG_PATHPREFIX/demo_validate \
-R %RS_NAME -G %RG_NAME -T %RT_NAME" \
-d mysrs
```

For more information on the optional property variables used above, see "How to Create a Demo Resource of Type ORCL.gds" on page 28.

3. **Bring the resource online.**

```
# clresourcegroup online -eM mysrg
# clresource status mysrs
=== Cluster Resources ===
Resource Name       Node Name      State        Status Message
-------------       ---------      -----        --------------
mysrs               node1          Online       Online - System-log is online
                    node2          Online       Online - System-log is online
```

The *mysrs* resource now reflects the state of the demo proxy application, the SMF system-log.

Note - A resource of type `ORCL.gds_proxy` requires that the `demo_proxy_prenet_start` extension property is used. All other extension properties are optional.

---

4. **Display the proxy interval for the *mysrs* resource.**

The `Proxy_interval` extension property determines how often the *mysrs* resource checks the status of the SMF system-log. The default is 30 seconds, and can be changed using the `clresource`(8CL) command.

```
# clresource show -p proxy_interval mysrs

=== Resources ===
Resource:          mysrs
--- Standard and extension properties ---
Proxy_interval     30
   Class:          extension
   Description:      Prenet_start proxy interval (seconds)
   Per-node:         False
   Type:           int
```

5. **Disable the SMF system-log service on one node and verify that the *mysrs* resource reflects the new state of the system-log.**

   Within 30 seconds after you issue the `svcadm disable` command, the state and status on node1 should change.

```
# svcadm disable system-log
# clresource status mysrs

=== Cluster Resources ===
Resource Name       Node Name      State        Status Message
-------------       ---------      -----        --------------
mysrs               node1          Offline      Offline - System-log is offline
                    node2          Online       Online - System-log is online
```

   After you create the demo resource of type `ORCL.gds_proxy`, you can make additional customizations to the resource. See "Additional `ORCL.gds_proxy` Extension Properties" on page 57.

Using the GDSv2 Extension Properties

This section contains information about the extension properties you can use with a resource of type `ORCL.gds` and `ORCL.gds_proxy`.

`ORCL.gds` *method*`_command` Extension Properties

The table below lists the `ORCL.gds` *method*`_command` extension properties. See "The *method*`_command` Sequence" on page 23 for more information.

Property Name	Required	Comments
`Boot_command`	No	Any UNIX command.
`Fini_command`	No	Any UNIX command.
`Init_command`	No	Any UNIX command.
`Start_command`	Yes	Any UNIX command.
`Stop_command`	Yes/ No	Any UNIX command. Required if `PMF_managed=FALSE`.
`Validate_command`	No	Any UNIX command.
`Probe_command`	No	Any UNIX command. If `PMF_managed=TRUE` is set, an internal probe is used.

`Boot_command` Property

The `Boot_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

`Fini_command` Property

The `Fini_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

`Init_command` Property

The `Init_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

`Start_command` Property

The `Start_command` is a required property and starts the application. This command must be a UNIX command with arguments that can be passed directly to a shell to start the application.

The application in this context can be any software application in a traditional sense, but it could also just be a UNIX command similar to either of the following lines:

`Start_command=`*path to start my software application*
`Start_command="`*/usr/bin/touch /var/tmp/myrs*`"`

Note - If the `Start_command` does not leave behind at least one process, then `PMF_managed=FALSE` must be set. See "`PMF_managed` Property" on page 41 for more information. Furthermore, if `PMF_managed=FALSE` is set, then the `Stop_command` property is also required.

Note - If the default `Wait_for_online=TRUE` is set, then the `Probe_command` is executed within the `Start` callback method to determine if the application is online. GDSv2 passes an argument to the `Probe_command` to indicate if the `Probe_command` is being called within the `Start` callback method or if the `Probe_command` is being called by the GDSv2 probe after the resource has started successfully and is now online.

Passing an argument to the `Probe_command` provides the ability to code different behavior within the `Probe_command` when the resource is being started or after the resource has been started and is now online.

That argument is passed as the last argument to `Probe_command` and can contain the values *gds_start* when the `Probe_command` is executed within the `Start` callback method or *gds_probe* when the `Probe_command` is executed after the resource has started successfully and is now online.

See the `/opt/ORCLscgds/demo/demo_probe` file for an example. Following is a snippet of code from `demo_probe` that assigns the last passed argument to the method variable:

```
#!/usr/bin/ksh
eval typeset -r method=\$$#
```

`Stop_command` Property

The `Stop_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

Note - If `PMF_managed=FALSE` is set, then the `Stop_command` property is a required property.

`Validate_command` Property

The `Validate_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

When a resource is created, GDSv2 passes all resource properties as arguments to the `Validate_command`. When a resource property is updated, GDSv2 passes just those properties that are being updated.

The `/opt/ORCLscgds/lib/gds_functions` file provides helper function `gds_opts()` to process those arguments as upper case KSH global variables. Property values are as defined.

See the `/opt/ORCLscgds/demo/demo_validate` file for an example. Following is a snippet of code from `demo_validate`:

```
#!/usr/bin/ksh
. /opt/ORCLscgds/lib/gds_functions
get_opts "$@"
```

Note - Additionally, the function `get_opts()` processes an argument that GDSv2 supplies that is not a resource property but instead reflects per-node status about `SUNW.HAStoragePlus` resources that are used by this resource.

The KSH global variable HASP returns the following status codes:

`SCDS_HASP_NO_RESOURCE` Indicates that the resource does not depend on a `SUNW.HAStoragePlus` resource.

`SCDS_HASP_ERR_CONFIG` Indicates that at least one of the `SUNW.HAStoragePlus` resources on which the resource depends is located in a different resource group.

`SCDS_HASP_NOT_ONLINE` Indicates that a `SUNW.HAStoragePlus` resource on which the resource depends is not online on any potential primary node.

`SCDS_HASP_ONLINE_NOT_LOCAL` Indicates that at least one `SUNW.HAStoragePlus` resource on which the resource depends is online, but on another node.

`SCDS_HASP_ONLINE_LOCAL` Indicates that all `SUNW.HAStoragePlus` resources on which the resource depends are online on the node.

The preceding status codes have precedence over each other in the order in which they appear. For example, if a `SUNW.HAStoragePlus` resource is not online and another `SUNW.HAStoragePlus` is online on a different node, the status code is set to `SCDS_HASP_NOT_ONLINE` rather than `SCDS_HASP_ONLINE_NOT_LOCAL`.

Furthermore, if the `SUNW.HAStoragePlus` resource is managing a global file system, then the per-node HASP resource will report `SCDS_HASP_ONLINE_LOCAL` on the node where the `SUNW.HAStoragePlus` resource is online and `SCDS_HASP_ONLINE_NOT_LOCAL` on the other nodes.

Additional `ORCL.gds` Extension Properties

The `ORCL.gds` resource type includes extension properties that affect how a resource of this type behaves. With the examples that follow, you must ensure that the resource group *myrg* has been created. If you need to create the resource group, use the following command:

```
# clresourcegroup create -p pathprefix=/opt/ORCLscgds/demo myrg
```

`Child_mon_level` Property

Note - If you use Oracle Solaris Cluster administration commands, you can use the `Child_mon_level` property. If you use Agent Builder, you cannot use this property.

This property provides control over the processes that are monitored through the Process Monitor Facility (PMF). This property denotes the level up to which the forked children processes are monitored. This property works like the `-C` argument to the `pmfadm` command. See the `pmfadm`(8) man page.

Omitting this property, or setting it to the default value of -1, has the same effect as omitting the `-C` option on the `pmfadm` command. The result is that all children and their descendants are monitored.

`Debug_gds` Property

The `Debug_gds` extension property is set to `FALSE` by default. This property is required by Oracle Solaris Cluster Development and support. It can be useful to understand the various call sequences that occur within GDSv2. If `Debug_gds=FALSE` is set, no GDSv2 internal debug messages are sent to the system-log. Consequently, if `Debug_gds=TRUE` is set, all internal `debug_messages` are sent to the system-log.

Perform the following steps to send debug messages to the system-log:

1. Send all GDSv2 internal debug messages to the system-log.

   # **`clresource set -p debug_gds=TRUE`** *myrs*

2. (Optional) To set `Debug_gds` as a per-node extension property, you can set it for one node or set different values for each node.

```
# clresource set -p debug_gds=false myrs
# clresource set -p "debug_gds{node1}"=true myrs
# clresource show -p debug_gds myrs

=== Resources ===
Resource:                                       myrs
  --- Standard and extension properties ---
  Debug_gds{node1}:                             TRUE
  Debug_gds{node2}:                             FALSE
    Class:                                          extension
    Description:                                    Debug GDS code only
    Per-node:                                       True
    Type:                                           boolean
```

`Debug_level` Property

The `Debug_level` extension property is set to 0 by default. This property is part of the housekeeping KSH functions that provide trace and debug message support. To use `Debug_level`, your *method*`_command` script must source `/opt/ORCLscgds/lib` and call the `debug_message()` function at least once within the script. The *${DEBUG}* variable can then be invoked to react to the `Debug_level` extension property.

The `/opt/ORCLscgds/demo/demo_start` script contains an example:

```
# . /opt/ORCLscgds/lib/gds_functions

get_opts "$@"
debug_message "Script: demo_start - Begin"
${DEBUG}
```

Use these guidelines to understand how `Debug_level` works:

- Setting `Debug_level=0` does not produce any trace output or debug messages.
- Setting `Debug_level=1` does not produce any trace output; however, reduced debug messages are written to the system-log.
- Setting `Debug_level=2` produces trace output and all debug messages are written to the system-log.
- Setting `Debug_level=3` produces all debug messages that are written to the `DEBUG_LOGFILE,/var/cluster/logs/DS/${RESOURCETYPE}/message_log.${RESOURCE}`.

Note - To enable debug messages to be written to the system-log, the `/etc/syslog.conf` file must be amended and the SMF system-log service restarted. For example: **`*.err;kern.debug;daemon.debug;mail.crit /var/adm/messages`**.

Perform the following steps to set up trace and debug messages:

1. Set the debug level for *myrs*.

   # **`clresource set -p Debug_level=2`** *myrs*

   ```
   node1 - RESOURCE=myrs
   node1 - RESOURCEGROUP=myrg
   node1 - RESOURCETYPE=ORCL.gds:1
   node1 - OPERATION=update
   node1 - Debug_level=2
   node2 - RESOURCE=myrs
   node2 - RESOURCEGROUP=myrg
   node2 - RESOURCETYPE=ORCL.gds:1
   node2 - OPERATION=update
   node2 - Debug_level=2
   ```

   Trace information is written to the console when the resource is enabled and disabled. Debug messages are written to the system-log. For example:

   ```
   Sep  4 07:28:43 node1 SC[ORCL.gds:1,myrg,myrs]: [ID 382926
        daemon.debug] debug_message - Script: demo_start - Begin
   Sep  4 07:28:43 node1 SC[ORCL.gds:1,myrg,myrs]: [ID 382926 daemon.debug]
        debug_message - Script: demo_start - hostname is lh1
   ```

```
Sep  4 07:28:43 node1 SC[ORCL.gds:1,myrg,myrs]: [ID 382926 daemon.debug]
     debug_message - Script: demo_start - End (0)
```

2. (Optional) To set `Debug_level` as a per-node extension property, you can set it for one node or set different values for each node.

   # **`clrs set -p "debug_level{node1}"=2 -p "debug_level{node2}"=0`** *myrs*

   ```
   node1 - RESOURCE=myrs
   node1 - RESOURCEGROUP=myrg
   node1 - RESOURCETYPE=ORCL.gds:1
   node1 - OPERATION=update
   ```

`Interpose_logical_hostname` Property

The `Interpose_logical_hostname` extension property is empty (`""`) by default. This property determines if a logical hostname should be interposed whenever a system call to retrieve the hostname is made. Interposing a logical hostname provides a mechanism to return a logical hostname whenever a system call is made to retrieve the hostname. For example, when the physical node name is node1 and a `hostname(1)` command is issued, then node1 is returned.

However, assume you have a logical hostname, lh1, which is plumbed and available on node1. By interposing all system calls to retrieve the hostname, it is then possible to return lh1 when a `hostname(1)` command is issued. Interposing a logical hostname within GDSv2 requires that a value be set for the `Interpose_logical_hostname` property. You must also define symbolic links on each Oracle Solaris Cluster node.

Perform the following steps to define symbolic links on each cluster node so that GDSv2 can interpose the logical hostname from a secure library:

1. For each cluster node, create a symbolic link.

   # **`ln -s /usr/cluster/lib/libschost.so.1 /usr/lib/secure/libschost.so.1`**

2. For each AMD64 cluster node, create a symbolic link.

   # **`ln -s /usr/cluster/lib/amd64/libschost.so.1 /usr/lib/secure/64/libschost.so.1`**

3. For each SPARC cluster node, create a symbolic link.

   # **`ln -s /usr/cluster/lib/sparcv9/libschost.so.1 /usr/lib/secure/64/libschost.so.1`**

After the `Interpose_logical_hostname` is set and the symbolic links are defined, the `Interpose_logical_hostname` value can be returned to your *method*`_command` whenever a system call is made to retrieve the hostname:

- If `PMF_managed=TRUE` is set, then the `Interpose_logical_hostname` is automatically available to your `Start_command` and `Probe_command`.

- If `PMF_managed=FALSE` is set, then the GDSv2 function `interpose_logical_hostname()` is available to retrieve the `Interpose_logical_hostname` value.

The GDSv2 function `interpose_logical_hostname()` can also be used by *method*`_command` entries other than the `Start_command` and `Probe_command`.

Perform the following steps to retrieve the hostname.

1. Disable or delete the resource *myrs*.

   a. Disable the resource *myrs*.

      # **`clresource disable`** *myrs*

   b. Delete the resource *myrs*.

      # **`clresource delete`** *myrs*

2. Create the resource.

```
# clresource create -g myrg -t ORCL.gds \
-p Start_command="%RG_PATHPREFIX/demo_start -R %RS_NAME -G %RG_NAME -T %RT_NAME" \
-p Stop_command="%RG_PATHPREFIX/demo_stop -R %RS_NAME -G %RG_NAME -T %RT_NAME" \
-p Probe_command="%RG_PATHPREFIX/demo_probe -R %RS_NAME -G %RG_NAME -T %RT_NAME" \
-p Validate_command="%RG_PATHPREFIX/demo_validate -R %RS_NAME -G %RG_NAME \
-T %RT_NAME" -d myrs
```

3. Interpose the logical hostname value of *lh1*.

Note - Ensure that a logical hostname is plumbed and available. See the `clreslogicalhostname`(8CL) man page for more information about creating a logical host.

   # **`clresource set -p PMF_managed=true -p interpose_logical_hostname=`***lh1 myrs*

   If `PMF_managed=TRUE` is set, appropriate environment variables are set to interpose the `Interpose_logical_hostname` value after the resource is enabled.

4. Enable the *myrs* resource.

   # **`clresource enable`** *myrs*

5. Determine the environment variables.

   # **`pmfadm -l ""`**

```
STATUS myrg,myrs,0.mon
pmfadm -c myrg,myrs,0.mon -n 4 -t 2 /bin/ksh -c '/opt/ORCLscgds/bin/gds_probe -R myrs
 -T ORCL.gds -G myrg'
        environment:
                LD_PRELOAD_32=/usr/lib/secure/libschost.so.1
```

```
                LD_PRELOAD_64=/usr/lib/secure/64/libschost.so.1
                SC_LHOSTNAME=lh1
        retries: 0
        owner: root
        monitor children: all
        pids: 4363
STATUS myrg,myrs,0.svc
pmfadm -c myrg,myrs,0.svc -a /usr/cluster/lib/sc/scds_pmf_action_script /bin/ksh -c
        '/usr/cluster/bin/hatimerun -t 299 /opt/ORCLscgds/demo/demo_start -R myrs -G
 myrg -T ORCL.gds ;
        echo $? > /var/cluster/run/tempgna4xi'
        environment:
                LD_PRELOAD_32=/usr/lib/secure/libschost.so.1
                LD_PRELOAD_64=/usr/lib/secure/64/libschost.so.1
                SC_LHOSTNAME=lh1
        retries: 0
        owner: root
        monitor children: all
        pids: 4313
#
```

If `PMF_managed=FALSE` is set, then the GDSv2 function `interpose_logical_hostname()` can be used to retrieve the Interpose_logical_hostname value.

An example of the GDSv2 function `interpose_logical_hostname()` is found in the `/opt/ORCLscgds/demo/demo_start` script. After `Interpose_logical_hostname=lh1` has been set for a resource, the following standalone program can also be used to set appropriate environment variables:

```
# /opt/ORCLscgds/bin/gds_libschost -R myrs -
G
myrg -T ORCL.gds:1

LD_PRELOAD_32=/usr/lib/secure/libschost.so.1
LD_PRELOAD_64=/usr/lib/secure/64/libschost.so.1
SC_LHOSTNAME=lh1
```

The GDSv2 function `interpose_logical_hostname()` uses the standalone program previously described in the `/opt/ORCLscgds/demo/demo_start` script.

`Num_probe_timeouts` Property

The `Num_probe_timeouts` extension property is set to `2` by default. This property determines when a complete failure should be returned by GDSv2.

In the example for `Timeout_delay`, a complete failure was alluded to whenever the `Probe_command` suffered a timeout. In this context, if the `Probe_command` suffers a timeout, the GDSv2 probe counts that as a failure. With `Num_probe_timeouts=2`, that failure is treated as a partial failure (two `Probe_command` timeouts are tolerated).

However, if the `Probe_command` suffers two successive timeouts, then that failure is treated as a complete failure. If `Num_probe_timeouts=5` is set, then five successive `Probe_command` timeouts must occur before a complete failure is returned by GDSv2. Likewise, if `Num_probe_timeouts=1` is set, then just one `Probe_command` timeout causes GDSv2 to return a complete failure.

When a complete failure is returned by GDSv2, the RGM queries the `Failover_mode` property to determine what action to take.

`PMF_managed` Property

The `PMF_managed` extension property is set to `TRUE` by default.

When this property is `TRUE`, the GDSv2 software ensures that the application is started under the control of the PMF. Consequently, when `PMF_managed=FALSE` is set, GDSv2 will not start the application under the control of the PMF.

Typically, an application that is under the control of the PMF must leave at least one process running after the application has been started. However, with `PMF_managed=FALSE`, it is possible to have an application that does not leave behind at least one process. For example, the application could simply create a file or amend another application's configuration and subsequently end without leaving behind at least one process.

Note - If `PMF_managed=FALSE` is set, then the `Stop_command` property is also required.

Perform the following steps to create a file for an application:

Note - The purpose of creating a file using a GDSv2 resource is simply to show that the *myrs* resource can be brought online without leaving behind at least one process. This feature can be quite powerful if the *myrs* resource is used as a dependent resource for other resources (for example, where you want the *myrs* resource to do something before other dependent resources are brought online).

---

1. Ensure that the file does not exist and disable or delete the GDSv2 resource *myrs*.
   a. Verify that the file does not exist.

      ```
      # ls -l /var/tmp/myrs
      ```

```
/var/tmp/myrs: No such file or directory
```

b. Disable the resource *myrs*.

```
# clresource disable myrs
```

c. Delete the resource *myrs*.

```
# clresource delete myrs
```

2. Create the resource *myrs*.

```
# clresource create -g myrg -t ORCL.gds \
-p Start_command="/bin/touch /var/tmp/myrs" \
-p Stop_command="/bin/rm -f /var/tmp/myrs" \
-p PMF_managed=false -d myrs
```

3. Enable the resource *myrs*, check its status, and verify that the file exists.

```
# clresource enable myrs
# clresource status myrs

=== Cluster Resources ===
Resource Name       Node Name     State        Status Message
-------------       ---------     -----        --------------
myrs                node1         Online       Online
                    node2         Offline      Offline

ls -l /var/tmp/myrs

rw-r--r--   1 root    root    0 Sept  2  04:07  /var/tmp/myrs
```

4. Disable the resource and verify that the file no longer exists.

```
# clresource disable myrs
ls -l /var/tmp/myrs

/var/tmp/myrs: No such file or directory
```

`Probe_command` Property

The `Probe_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

If `Probe_command` is set, then the GDSv2 probe will execute that command at intervals determined by the `Thorough_probe_interval` property and for the duration of the `Probe_timeout` property.

If `Probe_command` is not set and the default `PMF_managed=TRUE` is set, then an internal GDSv2 probe is used. This probe checks the application PMF tag to provide a faster application restart using PMF if all the application processes fail.

GDSv2 passes the following options and arguments to the `Probe_command`:

`-R` *rs* **`-G`** *rg* **`-T`** *rt* **`'gds_start | gds_probe'`**

The `/opt/ORCLscgds/lib/gds_functions` file provides the helper function `gds_opts()` to process the options and their arguments as upper case KSH global variables. Property values are as defined.

The last argument, `'gds_start | gds_probe'`, is provided so that you can code different behavior within the `Probe_command` when the resource is being started or after the resource has been started and is now online.

See the `/opt/ORCLscgds/demo/demo_probe` file for an example that captures the last argument into the method variable. That variable can then be used to perform any appropriate conditional processing. Following is a snippet of code from `demo_probe`:

```
#!/usr/bin/ksh
eval typeset -r method=\$$#
```

The `Probe_command` should return one of the following exit statuses, which is then processed by the GDSv2 probe:

0	Success. The application is working correctly.
100	Complete failure. The application is not working.
201	Immediate failover.

The RGM responds to a *Complete failure* or *Immediate failover* by checking the `Failover_mode` property. By default, `Failover_mode=SOFT` is set. See the `r_properties(7)` man page for more information.

With `Failover_mode=SOFT`, if a *Complete failure* is returned, GDSv2 will request a restart of the resource up to a maximum of the `Retry_count` property value within the time specified by the `Retry_interval` property.

If the number of restarts exceeds the value of `Retry_count` within the time specified by `Retry_interval`, GDSv2 will request a failover of the resource's group to another node.

With `Failover_mode=SOFT`, if an *Immediate failover* is returned, GDSv2 will request an immediate failover of the resource's group to another node.

It is also possible for the `Probe_command` to return cumulative failures to the GDSv2 probe as follows:

<100	Cumulative failure. The application is not completely working or not completely failed.

GDSv2 can process consecutive failures within the `Retry_interval`. For example, if the `Probe_command` returns 25 on consecutive occasions within the default `Retry_interval` of 370 seconds, then as soon as the cumulative failure reaches 100, a complete failure is declared. GDSv2 then responds to a complete failure as described above.

`Start_exit_on_error` Property

The `Start_exit_on_error` extension property is set to `FALSE` by default.

When this property is `FALSE`, the GDSv2 software attempts to continuously start the application within the `Start_timeout` period if the application fails to start.

When the `Start_exit_on_error` property is set to `TRUE`, the GDSv2 software will not attempt to continually start the application within the `Start_timeout` period.

This can be advantageous if the application is expected to start immediately on the first attempt. Consequently, if the application fails to start on the first attempt, a `Start_failed` error occurs, without waiting for the `Start_timeout` period to expire.

Note - The RGM reacts to a `Start_failed` error by checking the `Failover_mode` property. Consequently, if the default `Failover_mode=SOFT` is set, then the RGM attempts to fail over the resource group to another Oracle Solaris Cluster node.

Perform the following steps to attempt to start an application:

Note - The `Start_command` string below is expected to be successful after it is executed. However, the `Start_command` will only work on node2. Nevertheless, the purpose of this feature is to demonstrate the behavior of the `Start_exit_on_error` property.

---

1. Disable or delete the resource *myrs*.
   a. Disable the resource *myrs*.

      ```
      # clresource disable myrs
      ```

   b. Delete the resource *myrs*.

      ```
      # clresource delete myrs
      ```

2. Set the `Start_exit_on_error` property.

```
# clresource create -g myrg -t ORCL.gds \
-p Start_command="/bin/uname -n | /bin/grep node2" -p Start_exit_on_error=TRUE \
-p Stop_commend=/bin/true -p PMF_managed=false \
-d myrs
```

3. Enable the property.

```
# clresource enable myrs

clrs:  (C748634) Resource group myrg failed to start on chosen node and might
fail over to other node(s)

clresource status myrs

=== Cluster Resources ===
Resource Name       Node Name      State        Status Message
-------------       ---------      -----        --------------
myrs                node1          Offline      Offline
                    node2          Online       Online
```

Note - The `Start_command="/bin/uname -n | /bin/grep node2"` will only be successful on node2. The system-log on node1 contains the following:

```
Sep 2 04:59:45 node1 SC[,ORCL.gds:1,myrg,myrs,gds_start]:
     [ID 186822 daemon.error] /bin/uname -n | /bin/grep node1 has failed rc=1
Sep 2 04:59:45 node1 SC[,ORCL.gds:1,myrg,myrs,gds_start]:
     [ID 475178 daemon.notice]  Start_exit_on_error=true has been set. The
     resource will enter a start failed state.
```

However, the RGM reacts to a `Start_failed` error by querying the `Failover_mode` setting. Consequently, when `Failover_mode=SOFT` was set, the resource group failed over to node2, where the `Start_command` was successful. Because the `PMF_managed=FALSE` was also set, a `Stop_command` is required. In this scenario, it is acceptable to not invoke the `STOP` action by using `Stop_command=/bin/true`.

`Stop_exit_on_error` Property

The `Stop_exit_on_error` extension property is set to `FALSE` by default.

If `Stop_exit_on_error=TRUE`, `Stop_command`, and `PMF_managed=TRUE` were all set, then if the `Stop_command` property returns a non-zero exit status, the resource immediately enters a `Stop_failed` state. The GDSv2 software stops monitoring the process IDs running under the

PMF tag; however, the PMF tag will still exist. Some application process IDs might still be running under the PMF tag, but the PMF does not monitor those process IDs.

Consequently, setting the `Stop_exit_on_error=TRUE` property is only useful when you also have the `PMF_managed=TRUE` property set. In this scenario, `Stop_exit_on_error=TRUE` prevents the PMF from sending the `Stop_signal` to the process IDs running under the PMF tag. This might be useful to determine why the `Stop_command` property failed to stop the application (for example, before the GDSv2 application cleans up the process IDs running under the PMF tag).

For example, perform the following steps to stop the application:

1. Disable or delete the resource *myrs*.
   a. Disable the resource *myrs*.

      # clresource disable *myrs*
   b. Delete the resource *myrs*.

      # clresource delete *myrs*
2. Create the resource and set the `Stop_exit_on_error=TRUE` property.

   ```
   # clresource create -g myrg -t ORCL.gds \
   -p Start_command="/bin/sleep 1800 &" \
   -p Stop_command="/bin/false" \
   -p Stop_exit_on_error=true \
   -d myrs
   ```
3. Enable the resource and check its status.

   # **clresource enable** *myrs*

   # **clresource status** *myrs*

   ```
   === Cluster Resources ===
   Resource Name       Node Name      State        Status Message
   -------------       ---------      -----        --------------
   myrs                node1          Online       Online - Service is online.
                       node2          Offline      Offline
   ```
4. Disable the resource.

   # **clresource disable** *myrs*

   ```
   resource group in ERROR_STOP_FAILED state requires operator attention
   ```
5. Check the status of the resource.

   # **clresource status** *myrs*

   ```
   === Cluster Resources ===
   Resource Name       Node Name      State        Status Message
   ```

```
-------------       ---------      -----        --------------
myrs                node1          Stop_failed  Faulted
                    node2          Offline      Offline
```

6. Display the PMF tag for the *myrs* resource.

```
# pmfadm -l myrg,myrs,0.svc
pmfadm -c myrg,myrs,0.svc -a /usr/cluster/lib/sc/scds_pmf_action_script \
/bin/ksh -c \
'/usr/cluster/bin/hatimerun -t 299 /bin/sleep 1800 &; echo $? > \
/var/cluster/run/temp3PaWJC'

     retries:  0
     owner:  root
     monitor children: all
     pids:  14624 14626
```

When the *myrs* resource is disabled, the `Stop_command` is executed. However, `Stop_command=/bin/false` was set, thereby inducing a `Stop_failed` error. When `Stop_exit_on_error=TRUE` was set, the GDSv2 application exits immediately with a `Stop_failed` error and does not attempt to clean up the process IDs running under the PMF tag.

The system-log on node1 also contains the following information:

```
Sep  2 06:11:41 node1 SC[,ORCL.gds:1,myrg,myrs,gds_stop]:
     [ID 186822 daemon.error] /bin/false  has failed rc=255
Sep  2 06:11:41 node1 SC[,ORCL.gds:1,myrg,myrs,gds_stop]: [ID 943012
     daemon.error] Stop_exit_on_error=true has been set. The resource will enter
     a stop failed state.
Sep  2 06:11:41 node1 Cluster.RGM.global.rgmd: [ID 938318 daemon.error]
     Method <gds_stop> failed on resource <myrs> in resource group <myrg>
     [exit code<1>, time used: 0% if timeout <300 seconds>]
```

`Stop_signal` Property

This property specifies a value that identifies the signal to stop an application through the PMF. See the `signal.h`(3HEAD) man page for a list of the integer values that you can specify. The default value is 15 (SIGTERM).

`Timeout_delay` Property

The `Timeout_delay` extension property is set to `FALSE` by default. This extension property affects the GDSv2 probing algorithm and attempts to prevent a `Probe_command` timeout when the system is under a heavy load.

Note - The `Probe_command` is executed periodically by the GDSv2 program, `gds_probe`, to determine if the application is healthy. When the system is under a heavy load, the `Probe_command` might be stuck waiting to execute as other higher-priority workload is executing. For example, if `Probe_timeout=30` and `Timeout_delay=FALSE` are set and the system is under a heavy load, the `Probe_command` could suffer a probe timeout.

When this probe timeout occurs, the GDSv2 software is unable to tell if the application is healthy and might determine that a complete failure has occurred. If a complete failure is declared, the RGM queries the `Failover_mode` property to determine what action to take. However, if `Probe_timeout=30` and `Timeout_delay=TRUE` are set and the system is under load, the timer for `Probe_timeout` will be delayed until the `Probe_command` is actually executing (rather than just being scheduled to execute).

The GDSv2 probe executes the `Probe_command` under a timeout clock and uses the `fork(2)` and `exec(2)` calls to execute the `Probe_command` as a new process. On a heavily loaded system, there can be seconds of delay from the time that the child process is forked until the time that the child process is executing the `Probe_command`.

If `Timeout_delay`=FALSE is set, the timeout clock is started as soon as the child process is forked.

If `Timeout_delay`=TRUE is set, the timeout clock is started only when the child process has started to execute.

There are advantages to both settings and you should consider the impact of setting `Timeout_delay`.

If the system is heavily loaded you might want a probe timeout to occur so that the RGM can attempt an application recovery by querying the `Failover_mode` property. In this case, on a heavily loaded system setting `Timeout_delay`=FALSE would be appropriate and is the default setting.

If the system is heavily loaded and you want to guarantee that the timeout clock is started only when the `Probe_command` has started to execute, then setting `Timeout_delay`=TRUE would be appropriate. However, there is no guarantee that a probe timeout might not still occur. Instead, the timeout clock is just delayed until `Probe_command` has started to execute. If the `Probe_command` still struggles to complete, once the timeout clock has been started, then a probe timeout can still occur.

If a probe timeout occurs, a failure is returned to GDSv2. By default, `Num_probe_timeouts`=2 is set meaning that two consecutive probe timeouts will result in a complete failure. When a complete failure is returned by GDSv2, the RGM queries the `Failover_mode` property to determine what action to take.

There is no practical example to actively demonstrate `Timeout_delay`.

`Wait_for_online` Property

The `Wait_for_online` extension property is set to `TRUE` by default.

When this property is `TRUE`, the GDSv2 software executes the `Probe_command` within the `START` method for the duration of `Start_timeout` when the resource is being enabled.

Note - If the `Probe_command` is not set and `PMF_managed=TRUE` is set, a dummy probe is used for the `Probe_command`. This dummy probe simply checks if the associated PMF tag exists.

When the resource is being started (enabled), if the Probe_command returns a zero exit status, the application is deemed to be available and the resource then enters an Online state. If `Wait_for_online=FALSE` is set, the GDSv2 software does not attempt to execute the `Probe_command` within the `START` method. Instead, if the `Start_command` exits with a zero exit status, then the resource enters an Online state. Otherwise, the resource enters a `Start_failed` state.

The RGM queries the `Failover_mode` property to determine what action to take from a `Start_failed` state. This information can be useful when you do not want to wait for the `Probe_command` to declare a zero return code before the resource enters an Online state.

Perform the following steps to simulate an application that takes more than 10 seconds to start:

1. Disable or delete the resource *myrs*.

   a. Disable the resource *myrs*.

      ```
      # clresource disable myrs
      ```

   b. Delete the resource *myrs*.

      ```
      # clresource delete myrs
      ```

2. Create the following scripts on each Oracle Solaris Cluster node.

   ```
   # cat /var/tmp/start

   #!/usr/bin/ksh

   /var/tmp/start_child &
   exit 0

   # cat /var/tmp/start_child
   ```

```
#!/usr/bin/ksh

sleep 10
/bin/touch /var/tmp/myrs
exit 0
```

cat /var/tmp/probe

```
#!/usr/bin/ksh

if [[ -f /var/tmp/myrs ]]; then
     exit 0
else
     exit 100
fi
```

Note - Create each of these scripts in this procedure on each Oracle Solaris Cluster node. Ensure that these scripts can be executed.

The example above shows that the `/var/tmp/start` will execute a background job called `/var/tmp/start_child`. The `/var/tmp/start_child` sleeps for 10 seconds and then touches the `/var/tmp/myrs`. The `Start_command=/var/tmp/start` should then exit with a zero exit status.

Note - The purpose of `/var/tmp/start` and `/var/tmp/start_child` is to simulate an application that takes some time to start, such as 10 seconds. All the scripts described above should be created on every Oracle Solaris Cluster node. The `/var/tmp/probe` checks if the application is running and is used by the `Probe_command` below.

---

3. Create the *myrs* resource.

```
# clresource create -g myrg -t ORCL.gds \
-p Start_command=/var/tmp/start \
-p Stop_command="/bin/rm -f /var/tmp/myrs" \
-p Probe_command=/var/tmp/probe \
-p PMF_managed=false \
-d myrs
```

4. Enable the resource and check its status.

time clresource enable *myrs*

```
real   0m10.45s
```

```
user   0m0.07s
sys    0m0.03s
```

clresource status *myrs*

```
=== Cluster Resources ===
Resource Name   Node Name    State    Status Message
-------------   ---------    -----    --------------
myrs            node1        Online   Online - Service is online.
                node2        Offline  Offline
```

The following example shows how the *myrs* resource is created using the `Wait_for_online=FALSE` and immediately enters an Online state. However, the resource status is degraded because the `Probe_command` has not yet returned a zero exit status.

Perform the following steps to immediately put a resource into an online state and then into a degraded state:

1. Disable the resource *myrs*.

   # **clresource disable** *myrs*
2. Delete the resource *myrs*.

   # **clresource delete** *myrs*
3. Create the *myrg* resource.

   ```
   # clresource -g myrg -t ORCL.gds \
   -p Start_command=/var/tmp/start \
   -p Stop_command="/bin/rm -f /var/tmp/myrs" \
   -p Probe_command=/var/tmp/probe \
   -p PMF_managed=false \
   -p Wait_for_online=false -d myrs
   ```
4. Enable the resource and check its status.

   # **time clresource enable** *myrs*

   ```
   real    0m0.32s
   user    0m0.07s
   sys     0m0.03s
   ```

   # **clresource status** *myrs*

   ```
   === Cluster Resources ===
   Resource Name       Node Name      State        Status Message
   -------------       ---------      -----        --------------
   myrs                node1          Online       Degraded - Service is degraded.
                       node2          Offline      Offline
   ```

After 60 seconds, check the status of the file again.

clresource status *myrs*

The Probe_Command is executed periodically. After the Thorough_probe_interval (60 seconds), the Probe_command is executed again. This time the probe is successful and the resource status enters an Online status.

```
=== Cluster Resources ===
Resource Name       Node Name      State        Status Message
-------------       ---------      -----        --------------
myrs                node1          Online       Online - Service is online.
                    node2          Offline      Offline
```

Wait_probe_limit Property

The Wait_probe_limit extension property is set to 0 by default.

This extension property is used when Wait_for_online=TRUE is set. See "Wait_for_online Property" on page 49 for more information.

When Wait_for_online=TRUE is set, GDSv2 executes the Probe_command within the START method for the duration of Start_timeout or until the Probe_command returns a zero exit status. The Probe_command is attempted every two seconds.

By default, Start_timeout=300 is set and consequently the Probe_command could be attempted many times until it is successful.

Three possible scenarios could occur:

- Wait_probe_limit=0 – The Probe_command is attempted for the duration of Start_timeout, until the Probe_command returns a zero exit status. Otherwise, the Probe_command attempts will continue until the RGM declares a START timeout.
- Wait_probe_limit=1 – The Probe_command is attempted just once during processing of the Wait_for_online property. Likewise, if Wait_probe_limit=8 is set, then the Probe_command makes eight attempts during the Wait_for_online processing.
- Wait_probe_limit=2 – The following procedure illustrates a simple example of Wait_probe_limit=2. The same scripts were used here as in the Wait_for_online=TRUE example in the "Wait_for_online Property" on page 49 section. In the first example when the default Wait_for_online=TRUE was set, the clrs enable *myrs* command took approximately 10 seconds to complete. However, in the example below, the Wait_probe_limit=2 is set and the clresource enable *myrs* command takes approximately four seconds to complete.

Perform the following steps to attempt several times to start the resource:

1. Disable the resource *myrs*.

   # **clresource disable** *myrs*

2. Delete the resource *myrs*.

   # **clresource delete** *myrs*

3. Create the *myrg* resource group.

```
# clresource create -g myrg -t ORCL.gds \
-p Start_command=/var/tmp/start \
-p Stop_command="/bin/rm -f /var/tmp/myrs" \
-p Probe_command=/var/tmp/probe \
-p PMF_managed=false \
-p Wait_probe_limit=2 \
-d myrs
```

4. Enable the resource and check its status.

   # **time clresource enable** *myrs*

```
clrs:  (C748634) Resource group myrg failed to start on chosen node
and might fail over to other node(s)
real    0m4.795s
user    0m0.075s
sys     0m0.035s
```

   Check the resource status.

   # **clresource status** *myrs*

```
=== Cluster Resources ===
Resource Name      Node Name      State         Status Message
-------------      ---------      -----         --------------
myrs               node1          Offline       Offline
                   node2          Starting      Unknown - Starting
```

   Recheck the resource status.

   # **clresource status** *myrs*

```
=== Cluster Resources ===
Resource Name      Node Name      State         Status Message
-------------      ---------      -----         --------------
myrs               node1          Online        Online - Service is online.
                   node2          Offline       Offline
```

In the preceding procedure, the resource *myrs* is being enabled but fails after approximately four seconds (the `Wait_probe_limit=2` was set and the `Probe_command` is attempted every two seconds after the last attempt). Consequently, the `Probe_command` did not return a zero exit status within those two attempts. The GDSv2 software then returned a `START` failed and the RGM declared a `Start_failed` state.

However, `Failover_mode=SOFT` was set by default and the RGM then failed over the resource group from node1 to node2 (the first `clresource status` *myrs* command shows the resource *myrs* being started on node2). However, when starting on node2, the `Probe_command` again also failed to return a zero exit status within two `Wait_probe_limit` attempts. Consequently, the GDSv2 software again returned a `START` failed and the RGM declared a `Start_failed` state. Because of the `Failover_mode=SOFT` setting, a failover of the resource group from node2 to node1 is now attempted.

Note - The same scripts were used here as in the `Wait_for_online=TRUE` example in "`Wait_for_online` Property" on page 49. As such, the `/var/tmp/start` script executes the `/var/tmp/start_child` script in the background. That script sleeps for 10 seconds before touching the file (`/var/tmp/myrs`) that `Probe_command` is checking.

The first attempt to enable resource *myrs* on node1 took approximately four seconds, and even though you cannot see it on the terminal, the first attempt to enable resource *myrs* on node2 also took approximately four seconds. With the second attempt to start resource *myrs* on node1, `/var/tmp/start_child` had already consumed approximately eight seconds of its 10-second sleep. Consequently, with `Wait_probe_limit=2` set, the second attempt to start the resource *myrs* was successful and the resource entered an Online state.

The system-log on node1 and node2 contains the following messages:

```
Sep  3 00:44:13 node1 SC[,ORCL.gds:1,myrg,myrs,gds_start]: [ID 496934 daemon.notice]
     wait_probe_limit=2 is set, resource will enter a start failed state.
Sep  3 00:44:17 node2 SC[,ORCL.gds:1,myrg,myrs,gds_start]: [ID 496934 daemon.notice]
     wait_probe_limit=2 is set, resource will enter a start failed state.
```

`ORCL.gds_proxy` *method*`_command` Extension Properties

The table below lists the `ORCL.gds_proxy` *method*`_command` extension properties. See "The *method*`_command` Sequence" on page 23 for more information.

Property Name	Required	Comments
`Boot_command`	No	Any UNIX command.
`Init_command`	No	Any UNIX command.
`Fini_command`	No	Any UNIX command.
`Prenet_start_command`	Yes	Any UNIX command.
`Postnet_stop_command`	No	Any UNIX command.
`Validate_command`	No	Any UNIX command.

`Boot_command` Property

The `Boot_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

`Init_command` Property

The `Init_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

`Fini_command` Property

The `Fini_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

`Prenet_start_command` Property

The `Fini_command` is a required property and starts the proxy daemon. This command must be a UNIX command with arguments that can be passed directly to a shell to start the application.

`Postnet_stop_command` Property

The `Postnet_stop_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

`Validate_command` Property

The `Validate_command` is not a required property. If set, this command must be a UNIX command with arguments that can be passed directly to a shell.

When a resource is created, GDSv2 passes all resource properties as arguments to the `Validate_command`. When a resource property is updated, GDSv2 passes just those properties that are being updated.

The `/opt/ORCLscgds/lib/gds_functions` file provides helper function `gds_opts()` to process those arguments as upper case KSH global variables. Property values are as defined.

See the `/opt/ORCLscgds/demo/demo_validate` file for an example. The following is a snippet of code from `demo_validate`:

```
#!/usr/bin/ksh
. /opt/ORCLscgds/lib/gds_functions
get_opts "$@"
```

Note - Additionally, the function `get_opts()` processes an argument that GDSv2 supplies that is not a resource property but instead reflects per-node status about `SUNW.HAStoragePlus` resources that are used by this resource.

The KSH global variable HASP returns the following status codes:

`SCDS_HASP_NO_RESOURCE` Indicates that the resource does not depend on a `SUNW.HAStoragePlus` resource.

`SCDS_HASP_ERR_CONFIG` Indicates that at least one of the `SUNW.HAStoragePlus` resources on which the resource depends is located in a different resource group.

`SCDS_HASP_NOT_ONLINE` Indicates that a `SUNW.HAStoragePlus` resource on which the resource depends is not online on any potential primary node.

`SCDS_HASP_ONLINE_NOT_LOCAL` Indicates that at least one `SUNW.HAStoragePlus` resource on which the resource depends is online, but on another node.

`SCDS_HASP_ONLINE_LOCAL` Indicates that all `SUNW.HAStoragePlus` resources on which the resource depends are online on the node.

The preceding status codes have precedence over each other in the order in which they appear. For example, if a `SUNW.HAStoragePlus` resource is not online and another `SUNW.HAStoragePlus`

is online on a different node, the status code is set to `SCDS_HASP_NOT_ONLINE` rather than `SCDS_HASP_ONLINE_NOT_LOCAL`.

Furthermore, if the `SUNW.HAStoragePlus` resource is managing a global file system, then the per-node HASP resource will report `SCDS_HASP_ONLINE_LOCAL` on the node where the `SUNW.HAStoragePlus` resource is online and `SCDS_HASP_ONLINE_NOT_LOCAL` on the other nodes.

Additional `ORCL.gds_proxy` Extension Properties

The `ORCL.gds_proxy` resource type includes extension properties that affect how a resource of this type behaves. With the examples that follow, you must ensure that the resource group *mysrg* has been created. If not, create the resource group:

```
# clresourcegroup create -p pathprefix=/opt/ORCLscgds/demo -S mysrg
```

`Debug_gds` Property

See "`Debug_gds` Property" on page 36. If you use the examples from this section, change *myrs* to *mysrs* and *myrg* to *mysrg*.

`Debug_level` Property

See "`Debug_level` Property" on page 36. If you use the examples from this section, change *myrs* to *mysrs* and *myrg* to *mysrg*.

`Interpose_logical host` Property

See "`Interpose_logical_hostname` Property" on page 38. If you use the examples from this section, change *myrs* to *mysrs*.

`Stop_signal` Property

See "`Stop_signal` Property" on page 47. If you use the examples from this section, change *myrs* to *mysrs* and *myrg* to *mysrg*.

Using the GDSv2 Demo Scripts

This section contains information about the demo scripts that are provided with GDSv2. These demo scripts can be used with a resource of type `ORCL.gds` or `ORCL.gds_proxy` to start, stop, and monitor or proxy the demo applications.

The benefit of a demo application is to quickly deploy a GDSv2 resource with minimal effort. You can then experiment with the various GDSv2 extension properties to learn about the functionality.

Note - GDSv2 demo scripts are located in the `/opt/ORCLscgds/demo` directory and use the Korn Shell (KSH). All functions listed below are located within the `/opt/ORCLscgds/lib` directory.

`ORCL.gds` Demo Scripts

The following demo scripts have been provided for a resource of type `ORCL.gds`:

- `/opt/ORCLscgds/demo/demo_probe`
- `/opt/ORCLscgds/demo/demo_start`
- `/opt/ORCLscgds/demo/demo_stop`
- `/opt/ORCLscgds/demo/demo_validate`

Note - Within these demo scripts, the host name or interposed host name is output as a debug message to the system log. The purpose of this is to show that if the `Interpose_logical_hostname` extension property has been set, then the exported SC_LHOSTNAME variable value is returned as the interposed host name.

In the Oracle Solaris Cluster environment, an application might attempt to access the same host name after a failover or switchover. As a result, the failover or switchover fails because the name of the physical host changes after the failover or switchover. In such a scenario, the application data service can use the `Interpose_logical_hostname` to provide a logical host name to the application rather than a physical host name.

`Demo_start` Script

The `demo_start` script starts an application, which is a background sleep for 1800 seconds. Additionally, it prints out debug messages to the system log to show Begin and End messages and the hostname or interposed hostname.

```
01 #
```

```
02 # Copyright (c)2013, 2014, Oracle and/or its affiliates. All rights reserved.
03 #
04 #ident "@(#)demo_start.ksh 1.2     14/02/10"
05 #
06
07 . /opt/ORCLscgds/lib/gds_functions
08 get_opts "$@"
09
10 debug_message "Script: demo_start - Begin"
11 ${DEBUG}
12 trap 'debug_message "Script: demo_start - End (${rc})"' EXIT
13 trap 'errtrap "Script: demo_start" ${LINENO}; rc=1; exit 1' ERR
14
15 typeset -i rc=0
16 typeset ilh
17 typeset pmf
18
19 ilh=$(/usr/cluster/bin/scha_resource_get -O extension \
20     -R ${RESOURCE} -G ${RESOURCEGROUP} interpose_logical_hostname)
21
22 ilh=$(echo ${ilh} | /usr/xpg4/bin/awk '{print$2}')
23
24 pmf=$(/usr/cluster/bin/scha_resource_get -O extension \
25     -R ${RESOURCE}  -G ${RESOURCEGROUP} pmf_managed)
26
27 pmf=$(echo ${pmf} | /usr/xpg4/bin/awk '{print$2}')
28
29 if (( ${#ilh} != 0 )); then
30     if [[ ${pmf} != TRUE ]]; then
31          interpose_logical_hostname ${RESOURCE} ${RESOURCEGROUP}
32     fi
33 fi
34
35 debug_message "Script: demo_start - hostname is $(/usr/bin/hostname)"
36
37 /usr/bin/sleep 1800 &
38
39 if [[ -f ${DEBUG_LOGFILE} ]]; then
40     /usr/bin/printf "--- $(date) - rc=${rc} \n" >> ${DEBUG_LOGFILE}
41     /usr/bin/printf "Script: demo_start - hostname is $(/usr/bin/hostname) \n" >>
 ${DEBUG_LOGFILE}
42 fi
43
44 exit ${rc}
```

- Lines 07-09 – The function `get_opts` processes all the arguments that GDSv2 passes to `demo_start`. Those arguments are processed as upper case KSH variables. Property values are as defined. For example, `RESOURCE=myrs`.

- Lines 10-11 – The function `debug_message` is called to output a Begin debug message to the system log. Additionally, the ${DEBUG} variable is set. See "`Debug_level` Property" on page 36 for more information.
- Lines 12-13 – The KSH trap built-in command is used to output an End debug message to the system log whenever the script exists. Additionally, if a command returns a non-zero exit status the KSH fake signal ERR is trapped and the function errtrap is called. Function `errtrap` will output an error message to the system log that contains the script name, line number of the command that returned a non-zero exit status, and the exit status that was returned by that command.
- Lines 19-22 – The Oracle Solaris Cluster program `scha_resource_get` retrieves the `interpose_logical_hostname` extension property, which is saved into the variable ilh.
- Lines 24-27 – The Oracle Solaris Cluster program `scha_resource_get` retrieves the `pmf_managed` extension property which is saved into the variable pmf.
- Lines 29-33 – If the `interpose_logical_hostname` extension property was set and the `pmf_managed` extension property was not set to `TRUE`, then the function `interpose_logical_hostname` is called. However, if `interpose_logical_hostname` was set and `pmf_managed` was set to `TRUE`, then environment variables for SC_LHOSTNAME are defined. See "`Interpose_logical_hostname` Property" on page 38 for more information.

  If the function `interpose_logical_hostname` is called, then environment variables for SC_LHOSTNAME are defined.
- Line 35 – Output a debug message to the system log that contains the script name and value from the `hostname` command. If environment variables for SC_LHOSTNAME exist, then the value for SC_LHOSTNAME is output.
- Line 37 – Start the application. For example, sleep 1800 in the background.
- Lines 39-42 – If variable ${DEBUG_LOGFILE} is set, then output some `debug_messages` to that file. When function `debug_messages` was first called on line 10, ${DEBUG_LOGFILE} was set to `/var/cluster/logs/DS/`*RT*`/message_log.`*RS*.. *RT* equals `ORCL.gds` and *RS* equals your resource name.

`Demo_probe` Script

The `demo_probe` script checks if the application is running (for example, the background sleep for 1800 seconds). Additionally, it prints out debug messages to the system log to show Begin and End messages and the hostname or interposed hostname.

```
01 #
02 # Copyright (c) 2013, 2014, Oracle and/or its affiliates. All rights reserved.
03 #
04 #ident  "@(#)demo_probe.ksh 1.2     14/02/10"
05 #
```

```
06
07 . /opt/ORCLscgds/lib/gds_functions
08 get_opts "$@"
09
10 eval typeset -r method=\$$#
11 debug_message "Script: demo_probe - Begin"
12 ${DEBUG}
13 trap 'debug_message "Script: demo_probe - End (${rc})"' EXIT
14 trap 'errtrap "Script: demo_probe" ${LINENO}; rc=1; exit 1' ERR
15
16 typeset -i rc=0
17 typeset ilh
18 typeset pmf
19
20 ilh=$(/usr/cluster/bin/scha_resource_get -O extension \
21     -R ${RESOURCE} -G ${RESOURCEGROUP} interpose_logical_hostname)
22
23 ilh=$(echo ${ilh} | /usr/xpg4/bin/awk '{print$2}')
24
25 pmf=$(/usr/cluster/bin/scha_resource_get -O extension \
26     -R ${RESOURCE} -G ${RESOURCEGROUP} pmf_managed)
27
28 pmf=$(echo ${pmf} | /usr/xpg4/bin/awk '{print$2}')
29
30 if (( ${#ilh} != 0 )); then
31     if [[ ${pmf} != TRUE ]]; then
32         interpose_logical_hostname ${RESOURCE} ${RESOURCEGROUP}
33     fi
34 fi
35
36 debug_message "Script: demo_probe - hostname is $(/usr/bin/hostname)"
37
38 if /usr/bin/ps -u root -o pid,args -z $(/usr/bin/zonename) | /usr/xpg4/bin/grep -q
 "sleep
     1800"; then
39     # Return code 0 declares a success.
40     rc=0
41 else
42     # Return code 100 declares a complete failure.
43   rc=100
44 fi
45
46 if [[ -f ${DEBUG_LOGFILE} ]]; then
47     /usr/bin/printf "--- $(date) - rc=${rc} \n" >> ${DEBUG_LOGFILE}
48     /usr/bin/printf "Script: demo_probe - method name is ${method} \n" >>
 ${DEBUG_LOGFILE}
49     /usr/bin/printf "Script: demo_probe - hostname is $(/usr/bin/hostname) \n" >>
           ${DEBUG_LOGFILE}
```

```
50 fi
51
52 exit ${rc}
```

- Lines 07-36 – Apart from line 10, these lines are explained with the `demo_start` script.
- Line 10 – The last argument that GDSv2 passes to the `demo_probe` script is saved in the `method` variable.

Note - The last argument, `'gds_start | gds_probe'`, is provided so that you can code different behavior within the `Probe_command`.

- Lines 38-44 – A check is made to see if the application (for example, sleep 1800) is still running. If the sleep is still running, then the `demo_probe` script will exit 0. Otherwise, exit 100 will be sent to GDSv2 to declare a complete failure.

  The RGM responds to a complete failure by checking the `Failover_mode` property to determine what recovery action to take. See "`Probe_command` Property" on page 42 and the `r_properties(7)` man page for more information.
- Lines 46-47 – These lines are explained with the `demo_start` script.

`Demo_stop` Script

The demo_stop script stops the application (for example, the background sleep for 1800 seconds). Additionally, it prints out debug messages to the system log to show Begin and End messages and the hostname or interposed host name.

```
01 #
02 # Copyright (c) 2013, 2014, Oracle and/or its affiliates. All rights reserved.
03 #
04 #ident  "@(#)demo_stop.ksh 1.2     14/02/10"
05 #
06
07 . /opt/ORCLscgds/lib/gds_functions
08 get_opts "$@"
09
10 debug_message "Script: demo_stop - Begin"
11 ${DEBUG}
12 trap 'debug_message "Script: demo_stop - End (${rc})"' EXIT
13 trap 'errtrap "Script: demo_stop" ${LINENO}; rc=1; exit 1' ERR
14
15 typeset -i rc=0
16 typeset ilh
17 typeset pmf
```

```
18
19 ilh=$(/usr/cluster/bin/scha_resource_get -O extension \
20     -R ${RESOURCE} -G ${RESOURCEGROUP} interpose_logical_hostname)
21
22 ilh=$(echo ${ilh} | /usr/xpg4/bin/awk '{print$2}')
23
24 pmf=$(/usr/cluster/bin/scha_resource_get -O extension \
25     -R ${RESOURCE} -G ${RESOURCEGROUP} pmf_managed)
26
27 pmf=$(echo ${pmf} | /usr/xpg4/bin/awk '{print$2}')
28
29 if (( ${#ilh} != 0 )); then
30     if [[ ${pmf} != TRUE ]]; then
31         interpose_logical_hostname ${RESOURCE} ${RESOURCEGROUP}
32     fi
33 fi
34
35 debug_message "Script: demo_stop - hostname is $(/usr/bin/hostname)"
36
37 pid=$(/usr/bin/ps -u root -o pid,args -z $(/usr/bin/zonename) | \
38     /usr/xpg4/bin/grep "sleep 1800" | /usr/xpg4/bin/grep -v grep | \
39     /usr/xpg4/bin/awk '{print $1}')
40
41 if (( ${#pid} != 0 )); then
42     /usr/bin/kill -9 ${pid}
43 fi
44
45 if [[ -f ${DEBUG_LOGFILE} ]]; then
46     /usr/bin/printf "--- $(date) - rc=${rc} \n" >> ${DEBUG_LOGFILE}
47     /usr/bin/printf "Script: demo_stop - hostname is $(/usr/bin/hostname) \n" >>
 ${DEBUG_LOGFILE}
48 fi
49
50 exit ${rc}
```

- Lines 07-35 – These lines are explained with the `demo_start` script.
- Lines 37-42 – Find the process ID for the application started by the `demo_start` script (for example, 'sleep 1800') and then kill that process ID.
- Lines 45-49 – These lines are explained with the `demo_start` script.

`Demo_validate` Script

The `demo_validate` script validates extension properties used by a resource of type `ORCL.gds`. The function `get_opts` provides upper case KSH global variables. Property values are as defined (for example, `RESOURCE=myrs`). Additionally, the function `get_opts` will set the

HASP KSH global variable (for example, HASP=SCDS_HASP_NO_RESOURCE). See “Validate_command Property” on page 34 for more information.

```
01 #
02 # Copyright (c) 2013, 2014, Oracle and/or its affiliates. All rights reserved.
03 #
04 #ident  "@(#)demo_validate.ksh 1.2     14/02/10"
05 #
06
07 . /opt/ORCLscgds/lib/gds_functions
08 get_opts "$@"
09
10 debug_message "Script: demo_validate - Begin"
11 trap 'debug_message "Script: demo_validate - End (${rc})"' EXIT
12 trap 'errtrap "Script: demo_validate" ${LINENO}; rc=1; exit 1' ERR
13 typeset -i rc=0
14
15 exit ${rc}
```

- Lines 07-15 – These lines are explained with the `demo_start` script.

`ORCL.gds_proxy` Demo Scripts

The following demo scripts have been provided for a resource of type `ORCL.gds_proxy`:

- `/opt/ORCLscgds/demo/demo_proxy_prenet_start`
- `/opt/ORCLscgds/demo/demo_proxy_postnet_stop`
- `/opt/ORCLscgds/demo/demo_validate`

Note - The RGM will execute the `demo_proxy_prenet_start` script before any logical host network interfaces are configured up and execute `demo_proxy_postnet_stop` after any logical host network interface are configured down. Nevertheless, it is still possible to set the `interpose_logical_hostname` property, which will return the exported SC_LHOSTNAME variable value as the interposed host name even though that host name may not be configured up.

`Demo_proxy_prenet_start` Script

The `demo_proxy_prenet_start` script is executed as a daemon and checks the state of the system log. Additionally, it prints out debug messages to the system log to show Begin and End messages and the hostname or interposed host name.

```
01 #
```

```
02 # Copyright (c) 2013, 2014, Oracle and/or its affiliates. All rights reserved.
03 #
04 #ident  "@(#)demo_proxy_prenet_start.ksh 1.2     14/02/10"
05 #
06
07 . /opt/ORCLscgds/lib/gds_functions
08 get_opts "$@"
09
10 debug_message "Script: demo_prenet_start_proxy - Begin"
11 ${DEBUG}
12 trap 'debug_message "Script: demo_prenet_start_proxy - End (${rc})"' EXIT
13 trap 'errtrap "Script: demo_prenet_start_proxy" ${LINENO}; rc=1; exit 1' ERR
14
15 typeset -i rc=0
16 typeset -r scha_control=/usr/cluster/bin/scha_control
17 typeset -r set_status=/usr/cluster/bin/scha_resource_setstatus
18 typeset -r rs_get=/usr/cluster/bin/scha_resource_get
19 typeset status
20 typeset interval
21
22 debug_message "Script: demo_prenet_start_proxy - hostname is $(/usr/bin/hostname)"
23
24 if [[ -f ${DEBUG_LOGFILE} ]]; then
25     /usr/bin/printf "--- $(date) - rc=${rc} \n" >> ${DEBUG_LOGFILE}
26     printf "Script: demo_prenet_start_proxy - hostname is $(/usr/bin/hostname) \n" >>
           ${DEBUG_LOGFILE}
27 fi
28
29 interval=$(/usr/cluster/bin/scha_resource_get -O extension -R ${RESOURCE} -G
 ${RESOURCEGROUP}
     proxy_interval)
30 interval=$(echo ${interval} | /usr/xpg4/bin/awk '{print $2}')
31
32 while :
33 do
34     status=$(/usr/bin/svcs -Ho state system-log)
35
36     case ${status} in
37         disabled)   ${scha_control} -O CHANGE_STATE_OFFLINE -R ${RESOURCE} -G
 ${RESOURCEGROUP}
38                 ${set_status} -R ${RESOURCE} -G ${RESOURCEGROUP} -s OFFLINE -m
 "System-log is
                       offline"
39                 ;;
40       online   ${scha_control} -O CHANGE_STATE_ONLINE -R ${RESOURCE} -G
 ${RESOURCEGROUP}
41                 ${set_status} -R ${RESOURCE} -G ${RESOURCEGROUP} -s OK -m "System-log
 is online"
```

```
42                ;;
43       *)       ${scha_control} -O CHANGE_STATE_OFFLINE -R ${RESOURCE} -G
 ${RESOURCEGROUP}
44                ${set_status} -R ${RESOURCE} -G ${RESOURCEGROUP} -s DEGRADED -m
 "System-log is
                       degraded"
45                ;;
46     esac
47
48     sleep ${interval}
49 done
50
51 exit ${rc}
```

- Lines 07-09 – The function get_opts will process all the arguments that GDSv2 passes to `demo_proxy_prenet_start`. Those arguments are processed as upper case KSH variables. Property values are as defined (for example, `RESOURCE=mysrs`).
- Lines 10-11 – The function `debug_message` is called to output a Begin debug message to the system log. Additionally, the ${DEBUG} variable is set. See "`Debug_level` Property" on page 36 for more information.
- Lines 12-13 – KSH trap built-in command is used to output an End debug message to the system log whenever the script exists. Additionally, if a command returns a non-zero exit status the KSH fake signal ERR is trapped and the function errtrap is called. Function `errtrap` will output an error message to the system log that contains the script name, line number of the command that returned a non-zero exist status and the exit status that was returned by that command.
- Line 22 – Output a debug message to the system log that contains the script name and value from the hostname command. If environment variables for SC_LHOSTNAME exist, then the value for SC_LHOSTNAME is output.
- Lines 24-27 – If variable ${DEBUG_LOGFILE} is set, then output some `debug_messages` to that file. When function `debug_messages` was first called on line 10, ${DEBUG_LOGFILE} was set to `/var/cluster/logs/DS/`*RT*`/message_log/.`*RS*. *RT* equals `ORCL.gds_proxy` and *RS* equals your resource name.
- Lines 29-30 – The Oracle Solaris Cluster program `scha_resource_get` retrieves the `proxy_interval` extension property, which is saved into the variable interval.
- Lines 32-49 – Perform a while loop sleeping for the duration of ${interval} on every iteration. During each iteration, check the state of the system log using the `svcs(1)` command and reflect that state as an Oracle Solaris Cluster resource state and status.

`Demo_proxy_postnet_stop` Script

The `demo_proxy_postnet_stop` script is executed when the daemon that was started by `demo_proxy_prenet_start` is being stopped. Additionally, it prints out debug messages to the system log to show Begin and End messages and the hostname or interposed host name.

```
01 #
02 # Copyright (c) 2013, 2014, Oracle and/or its affiliates. All rights reserved.
03 #
04 #ident  "@(#)demo_proxy_postnet_stop.ksh 1.3     14/02/10"
05 #
06
07 . /opt/ORCLscgds/lib/gds_functions
08 get_opts "$@"
09
10 debug_message "Script: demo_postnet_stop_proxy - Begin"
11 ${DEBUG}
12 trap 'debug_message "Script: demo_postnet_stop_proxy - End (${rc})"' EXIT
13 trap 'errtrap "Script: demo_proxy_postnet_stop" ${LINENO}; rc=1; exit 1' ERR
14
15 typeset -i rc=0
16 typeset -r set_status=/usr/cluster/bin/scha_resource_setstatus
17
18 debug_message "Script: demo_postnet_stop_proxy - hostname is $(/usr/bin/hostname)"
19
20 ${set_status} -R ${RESOURCE} -G ${RESOURCEGROUP} -s OFFLINE
21
22 if [[ -f ${DEBUG_LOGFILE} ]]; then
23     /usr/bin/printf "--- $(date) - rc=${rc} \n" >> ${DEBUG_LOGFILE}
24     /usr/bin/printf "Script: demo_postnet_stop_proxy - hostname is $(/usr/bin/
hostname) \n" >>
            ${DEBUG_LOGFILE}
25 fi
26
27 exit ${rc}
```

- Lines 07-27 – All these lines are explained with the `demo_proxy_prenet_start` script.

Using Subclassed GDSv2 Resource Types

This section contains information about subclassing a GDSv2 resource type.

Reasons to Subclass GDSv2 Resource Types

When using a resource of type ORCL.gds or ORCL.gds_proxy, you cannot deploy new extension properties that might be required for your application. For example, if you require a user name to start or stop and probe your application, you will typically have to hard code that user name within your scripts. Instead, you could subclass the GDSv2 resource type and then create a new extension property within the subclassed resource type.

Note - If you subclass a GDSv2 resource type and add a new extension property to the RTR file and provide a default value for that property, be careful how you provide those default values. The following table provides some sample default values that might fit what you want to achieve:

DEFAULT ="";	Blank string entry
DEFAULT = "foo";	String entry
DEFAULT ="foo bar";	String entry with multiple entries
DEFAULT ="'foo bar *'";	String entry with multiple entries and special characters. Single quotes are enclosed by double quotes.
DEFAULT =2;	Integer value 2
DEFAULT =TRUE;	Boolean entry
DEFAULT ="NONE";	Enum entry

How to Subclass the ORCL.gds Resource Type

1. **On one cluster node, assume the root role.**

2. **Copy the ORCL.gds Resource Type Registration file.**

```
# cd /opt/ORCLscgds/etc
# cp ORCL.gds your path/my.gds
```

Note - For consistency, copy the RTR file on all nodes of the cluster.

3. **Edit the copied file** *your path/my.gds***.**

   Change the following entries to reflect your new resource type name:

```
RESOURCE_TYPE = “gds”;
```

```
VENDOR_ID = my;
#% SERVICE_NAME = “my.gds”;
```

Create a new extension property within the new Resource Type Registration file. For example, edit *your path*/*my.gds* and copy the `Boot_command` extension property and amend it the `Username` extension property.

```
{ PROPERTY = Username; EXTENSION; STRING; DEFAULT = ""; TUNABLE = AT_CREATION;
          DESCRIPTION = "Username for my application";
}
```

Note - For consistency, edit the RTR file on all nodes of the cluster.

Other extension properties can be created and copied from an existing extension property to meet your requirements. For example, the `Boot_command` extension property was copied to create the `Username` extension property above. However, the `TUNABLE` attribute was amended to use `AT_CREATION`. See the `property_attributes(7)` man page for more information about resource property attributes.

4. **Register and list the new Resource Type.**

```
# clresourcetype register -f your path/my.gds my.gds
```

5. **Create a resource of the new Resource Type.**

```
# clresourcegroup create newrg
# clresource create -g newrg -t my.gds \
-p Start_command=your start command \
-p Username=me -d newrs
```

6. **List the new extension property from your resource.**

```
# clresource show -p username newrs
```

You have now successfully subclassed the `ORCL.gds` resource type. Your new resource type [*my.gds*] will behave exactly as the `ORCL.gds` resource type, except that you have introduced a new extension property.

Note - To retrieve the contents of the `Username` extension property, use the `/usr/cluster/bin/scha_resource_get` program as shown in the demo scripts below.

```
root@node1:~# user=$(/usr/cluster/bin/scha_resource_get -O extension -R newrs -G newrg
 username |
     tail  -1)
root@node1:~# echo $user
```

```
me
root@node1:~#
```

How to Subclass the ORCL.gds_proxy Resource Type

The steps to subclass the ORCL.gds_proxy resource type are similar to the steps for subclassing the ORCL.gds resource type.

1. **On one cluster node, assume the root role.**

2. **Copy the ORCL.gds_proxy Resource Type Registration file.**

```
# cd /opt/ORCLscgds/etc
# cp ORCL.gds_proxy your path/my.gds_proxy
```

Note - For consistency, copy the RTR file on all nodes of the cluster.

3. **Edit the copied file *your path/my.gds_proxy*.**

Change the following entries to reflect your new resource type name:

```
RESOURCE_TYPE = “gds_proxy”;
VENDOR_ID = my;
#% SERVICE_NAME = “my.gds_proxy”;
```

Create a new extension property within the new Resource Type Registration file. For example, edit *your path/my.gds_proxy* and copy the Boot_command extension property and amend it as the Username extension property.

```
{ PROPERTY = Username; EXTENSION; STRING; DEFAULT = ""; TUNABLE = AT_CREATION;
         DESCRIPTION = "Username for my application";
}
```

Note - For consistency, edit the RTR file on all nodes of the cluster.

Other extension properties can be created and copied from an existing extension property to meet your requirements. For example, the Boot_command extension property was copied to create the Username extension property. However, the TUNABLE attribute was amended to use AT_CREATION. See the property_attributes(7) man page for more information about resource property attributes.

4. **Register and list the new Resource Type.**

```
# clresourcetype register -f your path/my.gds_proxy  my.gds_proxy
```

5. **Create a resource of the new Resource Type.**

```
# clresourcegroup create -S newsrg
# clresource create -g newsrg -t my.gds_proxy \
-p Prenet_start_command=your prenet_start command \
-p Username=me -d newsrs
```

6. **List the new extension property from your resource.**

```
# clresource show -p username newsrs
```

You have now successfully subclassed the ORCL.gds_proxy resource type. Your new resource type [*my.gds_proxy*] will behave exactly as the ORCL.gds_proxy resource type, except that you have introduced a new extension property.

Note - To retrieve the contents of the Username extension property, use the /usr/cluster/bin/scha_resource_get program as shown in the demo scripts below.

```
root@node1:~# user=$(/usr/cluster/bin/scha_resource_get -O extension -R newsrs -G newsrg
 username |
            tail  -1)
root@node1:~# echo $user
me
root@node1:~#
```

Upgrading the ORCL.gds and ORCL.gds_proxy Resource Types

Upgrade the ORCL.gds and ORCL.gds_proxy resource types if the following conditions apply:

- If you update from Oracle Solaris Cluster 4.2 to Oracle Solaris 4.3 with Oracle Solaris Cluster 4.2 registered GDSv2 resources and want to use Debug_level=3, then you will also need to migrate the GDSv2 resources to the new version.
- You upgrade the ORCL.gds and ORCL.gds_proxy resource types to the latest version of Oracle Solaris Cluster from an earlier version of the data service.
- You update from an earlier version of the operating system.

For general instructions that explain how to upgrade a resource type, see "Upgrading a Resource Type" in *Planning and Administering Data Services for Oracle Solaris Cluster 4.4*. The information that you require to complete the upgrade of the resource type is provided in the subsections that follow.

Information for Registering the New Resource Type Version

The release of Oracle Solaris Cluster data services indicates the release in which the version of the resource type was introduced.

To determine the version of the resource type that is registered, use the `clresourcetype list` command.

For example:

```
# clrt list | grep ORCL.gds
ORCL.gds_proxy:1
ORCL.gds:1
ORCL.gds:2
ORCL.gds_proxy:2
```

Information for Migrating Existing Instances of the Resource Type

The information that you require to edit each instance of the resource type is as follows:

- You can perform the migration at anytime. It is *not* required that you disable or unmanage the resource before performing the migration.
- For Oracle Solaris Cluster 4.3, the required value of the `Type_version` property is `2`.

▼ How to Migrate Instances of GDSv2 Resource Type

1. **Register the new GDSv2 resource type.**

   For ORCL.gds:
   ```
   # clresource register ORCL.gds
   ```

   For ORCL.gds_proxy:
   ```
   # clresource register ORCL.gds_proxy
   ```

2. **Migrate the existing GDSv2 resources to the new version of GDSv2.**

```
# clresource set -p Type_version=2 resource
```

If Debug_level=3 is needed:

```
# clresource set -p Debug_level=3 resource
```

♦ ♦ ♦ CHAPTER 3

Using Agent Builder to Create a Service That Uses GDS or GDSv2

You can use Agent Builder to create the service that uses the GDS. Agent Builder is described in more detail in Chapter 9, “Oracle Solaris Cluster Agent Builder” in *Developing Data Services*.

This chapter covers the following topics:

Creating and Configuring GDS-Based Scripts

▼ How to Start Agent Builder and Create the Scripts

1. **Become an administrator that provides `solaris.cluster.modify` authorization.**
2. **Start Agent Builder.**

   ```
   # /usr/cluster/bin/scdsbuilder
   ```

3. **Type the vendor name.**
4. **Type the application name.**

   Note - The combination of vendor name and application name is used as the name of the package for the scripts.

5. **Go to the working directory.**

You can use the Browse drop-down menu to select the directory rather than typing the path.

6. **Select whether the data service is scalable or failover.**

7. **Select GDS or GDSv2.**

Note - If you select GDSv2, you can optionally choose to select proxy or subclass:

- You selected GDSv2, but not proxy or subclass. A resource of type `ORCL.gds` will be created.
- You selected GDSv2 and proxy, but not subclass. A resource of type `ORCL.gds_proxy` will be created.
- You selected GDSv2 and subclass, but not proxy. A resource of a new subclassed `ORCL.gds` will be created.
- You selected GDSv2, proxy, and subclass. A resource of a new subclassed `ORCL.gds_proxy` will be created.

---

8. **(Optional) Change the RT Version from the default value that is shown.**

Note - You cannot use the following characters in the RT Version field: space, tab, slash (`/`), backslash (`\`), asterisk (`*`), question mark (`?`), comma (`,`), semicolon (`;`), left square bracket (`[`), or right square bracket (`]`).

---

9. **Click Create.**
   Agent Builder creates the scripts. The results are displayed in the Output Log area.
   Note that the Create button is grayed out. You can now configure the scripts.

10. **Click Next.**
    The Configure screen appears.

▼ How to Configure the Scripts for GDS

After creating the scripts, you need to configure the new service.

1. **Type the location of the `start` command, or click Browse to locate the `start` command.**
   You can specify property variables. Property variables are described in "Using Property Variables" in *Developing Data Services*.

2. **(Optional) Type the location of the `stop` command, or click Browse to locate the `stop` command.**

   You can specify property variables. Property variables are described in "Using Property Variables" in *Developing Data Services*.

3. **(Optional) Type the location of the `validate` command, or click Browse to locate the `validate` command.**

   You can specify property variables. Property variables are described in "Using Property Variables" in *Developing Data Services*.

4. **(Optional) Type the location of the `probe` command, or click Browse to locate the `probe` command.**

   You can specify property variables. Property variables are described in "Using Property Variables" in *Developing Data Services*.

5. **(Optional) Specify new timeout values for the `start`, `stop`, `validate`, and `probe` commands.**

6. **Click Configure.**

   Agent Builder configures the scripts.

   Note - Agent Builder concatenates the vendor name and the application name to create the package name.

   A package for scripts is created and placed in the following directory:

   working-dir`/`*vendor-name-application*`/pkg`

   For example, `/export/wdir/NETapp/pkg`.

   Go to "How to Install the Generated Package" on page 79.

▼ How to Configure the Scripts for GDSv2 Non-proxy or Subclassed GDSv2 Non-proxy

After creating the scripts, you need to configure the new service.

1. **Type the location of the `start` command, or click Browse to locate the `start` command.**

You can specify property variables %RS_NAME, %RG_NAME, or %RT_NAME. See Chapter 2, “Creating a Data Service with GDSv2” for more information on property variables.

2. **(Optional) Type the location of the `stop` command, or click Browse to locate the `stop` command.**

   You can specify property variables %RS_NAME, %RG_NAME, or %RT_NAME. See Chapter 2, “Creating a Data Service with GDSv2” for more information on property variables.

3. **(Optional) Type the location of the `validate` command, or click Browse to locate the `validate` command.**

   You can specify property variables %RS_NAME, %RG_NAME, or %RT_NAME. See Chapter 2, “Creating a Data Service with GDSv2” for more information on property variables.

4. **(Optional) Type the location of the `probe` command, or click Browse to locate the `probe` command.**

   You can specify property variables %RS_NAME, %RG_NAME, or %RT_NAME. See Chapter 2, “Creating a Data Service with GDSv2” for more information on property variables.

5. **(Optional) Type the `interpose_logical_hostname` entry.**

6. **(Optional) Select the Disable PMF entry.**

   Selecting Disable PMF ensures that `PMF_managed=FALSE` is set. See Chapter 2, “Creating a Data Service with GDSv2” for more information on `PMF_managed`.

7. **(Optional) Specify new timeout values for the `start`, `stop`, `validate`, and `probe` commands.**

8. **Click Configure.**

   Go to “How to Install the Generated Package” on page 79.

▼ How to Configure Scripts for a GDSv2 Proxy or Subclassed GDSv2 Proxy

After creating the scripts, you will configure the new service.

1. **Type the location of the `Prenet_start` command, or click Browse to locate the `Prenet_start` command.**

You can specify property variables %RS_NAME, %RG_NAME, or %RT_NAME. See Chapter 2, “Creating a Data Service with GDSv2” for more information on property variables.

2. **(Optional) Type the location of the `Postnet_stop` command, or click Browse to locate the `Postnet_stop` command.**

   You can specify property variables %RS_NAME, %RG_NAME, or %RT_NAME. See Chapter 2, “Creating a Data Service with GDSv2” for more information on property variables.

3. **(Optional) Type the location of the `validate` command, or click Browse to locate the `Validate` command.**

   You can specify property variables %RS_NAME, %RG_NAME, or %RT_NAME. See Chapter 2, “Creating a Data Service with GDSv2” for more information on property variables.

4. **(Optional) Specify new timeout values for the `Prenet_start`, `Postnet_stop`, and `Validate` commands.**

5. **Click Configure.**

   Proceed to “How to Install the Generated Package” on page 79.

▼ How to Install the Generated Package

1. **On each node of the cluster, become an administrator that provides `solaris.cluster.modify` authorization.**

2. **On each node of the cluster, install the completed package.**

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

Note - This instruction applies to the SVR4 package that Agent Builder creates. If you need an IPS version of the package, use the `pkgsend` command to convert your SVR4 agent package to an IPS package, and use the `pkg add` command to install the IPS package. For more information, see the `pkgsend(1)` and `pkg(1)` man pages.

The following files are installed by `pkgadd`:

```
/opt/NETapp
/opt/NETapp/README.app
/opt/NETapp/man
/opt/NETapp/man/man1m
```

```
/opt/NETapp/man/man1m/removeapp.1m
/opt/NETapp/man/man1m/startapp.1m
/opt/NETapp/man/man1m/stopapp.1m
/opt/NETapp/man/man1m/app_config.1m
/opt/NETapp/util
/opt/NETapp/util/removeapp
/opt/NETapp/util/startapp
/opt/NETapp/util/stopapp
/opt/NETapp/util/app_config
```

Note - The man pages and script names correspond to the application name that you typed previously on the Create screen, preceded by the script name (for example, `startapp`).

3. On one node of the cluster, configure the resources and start the application.

```
# /opt/NETapp/util/startapp -h logicalhostname -p port-and-protocol-list
```

The arguments to the `startapp` script vary according to the type of resource: failover or scalable.

Note - To determine the command line that you need to type, check the customized man page, or run the `startapp` script without any arguments to display a usage statement.

To view the man pages, you need to specify the path to the man page. For example, to view the `startapp`(1M) man page, type:

```
# man -M /opt/NETapp/man startapp
```

To display a usage statement, type:

```
# /opt/NETapp/util/startapp
The resource name of LogicalHostname or SharedAddress must be
specified. For failover services:
Usage: startapp -h logicalhostname
-p port-and-protocol-list
[-n ipmpgroup-adapter-list]
For scalable services:
Usage: startapp -h shared-address-name
-p port-and-protocol-list
[-l load-balancing-policy]
[-n ipmpgroup/adapter-list]
[-w load-balancing-weights]
```

Output From Agent Builder

Agent Builder generates three scripts and a configuration file based on input that you provide when you create the package. The configuration file specifies the names of the resource group and the resource type.

The scripts are as follows:

- **Start script** – Configures the resources and starts the application that is under RGM control.
- **Stop script** – Stops the application and takes down resources and resource groups.
- **Remove script** – Removes the resources and resource groups that are created by the start script.

These scripts have the same interface and behavior as the utility scripts that are generated by Agent Builder for non-GDS-based data services. The scripts are put in an Oracle Solaris package that you can reuse across multiple clusters.

You can customize the configuration file to provide your own names for resource groups or other arguments that are normally given as arguments to the `clresource` and `clresourcegroup` commands. If you do not customize the scripts, Agent Builder provides default values for these arguments.

Command-Line Interface for Agent Builder

Agent Builder incorporates a command-line interface that provides the same functionality that the GUI provides. This interface consists of the commands `scdscreate` and `scdsconfig`. See the `scdscreate`(8HA) and `scdsconfig`(8HA) man pages for more information.

▼ How to Use the Command-Line Version of Agent Builder to Create a Service That Uses GDS

This section describes how to use the command-line interface to perform the same set of steps shown earlier in this chapter.

1. **Become an administrator that provides `solaris.cluster.modify` authorization.**
2. **Create the service by performing one of the following steps.**

- Create a failover service.

  ```
  # scdscreate -g -V NET -T app -d /export/wdir
  ```

- Create a scalable service.

  ```
  # scdscreate -g -s -V NET -T app -d /export/wdir
  ```

Note - The -d argument is optional. If you do not specify this argument, the current directory becomes the working directory.

3. Configure the service.

```
# scdsconfig -s "/export/app/bin/start" \
-e "/export/app/bin/configtest" \
-t "/export/app/bin/stop" \
-m "/export/app/bin/probe" -d /export/wdir
```

You can specify property variables. Property variables are described in "Using Property Variables" in *Developing Data Services*.

Note - Only the start command (`scdsconfig -s`) is required. All other options and arguments are optional.

4. On each node of the cluster, install the completed package.

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

Note - This instruction applies to the SVR4 package that Agent Builder creates. If you need an IPS version of the package, use the `pkgsend` command to convert your SVR4 agent package to an IPS package, and use the `pkg add` command to install the IPS package. For more information, see the `pkgsend(1)` and `pkg(1)` man pages.

The following files are installed by `pkgadd`:

```
/opt/NETapp
/opt/NETapp/README.app
/opt/NETapp/man
/opt/NETapp/man/man1m
/opt/NETapp/man/man1m/removeapp.1m
/opt/NETapp/man/man1m/startapp.1m
/opt/NETapp/man/man1m/stopapp.1m
/opt/NETapp/man/man1m/app_config.1m
/opt/NETapp/util
```

```
/opt/NETapp/util/removeapp
/opt/NETapp/util/startapp
/opt/NETapp/util/stopapp
/opt/NETapp/util/app_config
```

Note - The man pages and script names correspond to the application name that you typed previously on the Create screen, preceded by the script name (for example, `startapp`).

---

5. On one node of the cluster, configure the resources and start the application.

```
# /opt/NETapp/util/startapp -h logicalhostname -p port-and-protocol-list
```

The arguments to the `startapp` script vary according to the type of resource: failover or scalable.

Note - To determine the command line that you need to type, check the customized man page or run the `startapp` script without any arguments to display a usage statement.

To view the man pages, you need to specify the path to the man page. For example, to view the `startapp`(1M) man page, type:

```
# man -M /opt/NETapp/man startapp
```

To display a usage statement, type:

```
# /opt/NETapp/util/startapp
The resource name of LogicalHostname or SharedAddress must be specified.
For failover services:
Usage: startapp -h logicalhostname
-p port-and-protocol-list
[-n ipmpgroup/adapter-list]
For scalable services:
Usage: startapp -h shared-address-name
-p port-and-protocol-list
[-l load-balancing-policy]
[-n ipmpgroup/adapter-list]
[-w load-balancing-weights]
```

How to Use the Command-Line Version of Agent Builder to Create a Service That Uses GDS or a Subclassed GDSv2

This section describes how to use the command-line interface to perform the same set of steps shown earlier in this chapter.

1. **Become an administrator that provides `solaris.cluster.modify` authorization.**

2. **Create the service by performing one of the following steps.**

   - Create a failover service.

     ```
     # scdscreate -G -V vendor -T app appname \
     [-d working directory] \
     [-c] [-p]
     ```

   - Create a scalable service.

     ```
     # scdscreate -G -s -V vendor -T app appname \
     [-d working directory] \
     [-c] [-p]
     ```

   Use the following guidelines:

   - The `-d` argument is optional. If you do not specify this argument, the current directory becomes the working directory.
   - The `-c` argument is optional. If set, a subclassed GDSv2 resource type is created.
   - The `-p` argument is optional. If set, a proxy GDSv2 resource type is created.

   Note - If the `-c` argument is selected and the `-p` argument is not selected, then a subclassed `ORCL.gds` resource type is created. If the `-c` and `-p` arguments are selected, then a subclassed `ORCL.gds_proxy` resource type is created.

3. **Configure the service.**

   - For a non-proxy service, type:

     ```
     # scdsconfig -s "path to your start command" \
     [-d working directory] \
     [-e "path to your validate command"] \
     [-t "path to your stop command"] \
     [-m "path to your probe command"] \
     ```

```
[-l "interpose logical hostname"] \
-p
```

- For a proxy service, type:

```
# scdsconfig -s "path to your prenet_start command" \
[-d working directory] \
[-e "path to your validate command"] \
[-t "path to your postnet_stop command"] \
[-l "interpose logical hostname"]
```

Use the following guidelines:

- The `-s` argument is required. You should specify the path to your `start` or `prenet_start` command.
- The `-d` argument is optional. If you do not specify this argument, the current directory becomes the working directory.
- The `-e` argument is optional. If you specify this argument, you should specify the path to your validate command.
- The `-t` argument is optional. If you specify this argument, you should specify the path to your `stop` or `postnet_stop` command.
- The `-m` argument is optional. If you specify this argument, you should specify the path to your `probe` command.
- The `-l` argument is optional. If you specify this argument, you should specify the interpose logical host name.
- The `-p` argument is optional. Selecting this argument ensures that `PMF_managed=FALSE` is set. See Chapter 2, “Creating a Data Service with GDSv2” for more information on the `PMF_managed` command.

4. On each node of the cluster, install the completed package.

```
# cd working directory/pkg
# pkgadd -d . vendorappname
```

Note - This instruction applies to the SVR4 package that Agent Builder creates. If you need an IPS version of the package, use the `pkgsend` command to convert your SVR4 agent package to an IPS package, and use the `pkg add` command to install the IPS package. For more information, see the `pkgsend(1)` and `pkg(1)` man pages.

---

5. On one node of the cluster, configure the resources and start the application.

```
# cd /opt/vendorappname/util/startapp [arguments] logicalhostname -p port-and-protocol-list
```

Note - The arguments to the `startapp` script vary according to the type of resource you created and configured. To determine the command line that you need to type, check the customized man page or run the `startapp` script without any arguments to display a usage statement.

To view the man pages, you need to specify the path to the man page. For example, to view the `startapp(1M)` man page, type:

```
# man -M /opt/vendorappname/man startapp
```

Index

S

T

U