Sun Cluster Reference Manual for Solaris OS

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–0582–10
August 2005, Revision A

050505@9999

Contents

Preface

The *Sun Cluster Reference Manual* provides reference information for commands, functions, and other public interfaces in Sun™ Cluster software. This book is intended for experienced system administrators with extensive knowledge of Sun software and hardware. This book is not to be used as a planning or presales guide. The information in this book assumes knowledge of the Solaris™ Operating System and expertise with the volume manager software that is used with Sun Cluster software.

Both novice users and those familiar with the Solaris Operating System can use online man pages to obtain information about their SPARC™ based system or x86 based system and its features.

A man page is intended to answer concisely the question "What does this command do?" The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Note – Sun Cluster software runs on two platforms, SPARC and x86. The information in this book pertains to both platforms unless otherwise specified in a special chapter, section, note, bulleted item, figure, table, or example.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.

- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX® system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 6 contains available games and demos.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules, and the STREAMS-generic set of system calls are also described.
- Section 9 provides reference information that is needed to write device drivers in the kernel environment. This section describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures that drivers use to share information between the driver and the kernel.

The following is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if no bugs can be reported, no BUGS section is included. See the `intro` pages for more information and detail about each section, and `man`(1) for general information about man pages.

NAME — This section gives the names of the commands or functions that are documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. If a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single-letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] Brackets. The option or argument that is enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . .	Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "`filename . . .`".
|	Separator. Only one of the arguments separated by this character can be specified at a time.
{ }	Braces. The options and/or arguments enclosed within braces are interdependent. All characters within braces must be treated as a unit.

PROTOCOL	This section occurs only in subsection 3R and indicates the protocol description file.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. DESCRIPTION does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.
IOCTL	This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the `ioctl`(2) system call is called `ioctl` and generates its own heading. `ioctl` calls for a specific device are listed alphabetically (on the man page for that specific device). `ioctl` calls are used for a particular class of devices. All these calls have an `io` ending, such as `mtio`(7I).
OPTIONS	This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.

RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions that are declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` that indicates why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.
USAGE	This section lists special rules, features, and commands that require in-depth explanations. The subsections that are listed here are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible, a complete example, which includes command-line entry and machine response, is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero are returned for various error conditions.
FILES	This section lists all file names that are referred to by the man page, files of interest, and files created or required by commands. Each file name is followed by a descriptive summary or explanation.
ATTRIBUTES	This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes`(5) for more information.
SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition that caused the error.
WARNINGS	This section lists warnings about special conditions that could seriously affect your working conditions. WARNINGS is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. NOTES covers points of special interest to the user. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

SC31 1ha

NAME

rt_callbacks – callback interface for management of services as Sun Cluster resources

SYNOPSIS

rt_callbacks *method-path* -R *resource* -T *type* -G *group*

rt_callbacks *validate-path* [-c | -u] -R *resource* -T *type* -G *group* [-r *prop=val*] [-x *prop=val*] [-g *prop=val*]

DESCRIPTION

rt_callbacks, the callback interface for Sun Cluster resource types, defines the interface used by the cluster's Resource Group Manager (RGM) facility to control services as cluster resources. This man page describes the callback methods and arguments for the Version 2 API shipped with Sun Cluster 3.*x*.

The implementor of a resource type provides programs or scripts that serve as the callback methods:

method-path — The path the program that has been declared in the rt_reg(4) registration file, and registered with scrgadm(1M) as one of a resource type's callback methods: START, STOP, INIT, FINI, BOOT, PRENET_START, POSTNET_STOP, MONITOR_START, MONITOR_STOP, MONITOR_CHECK, or UPDATE.

validate-path — The path to the program that has been declared as a resource type's VALIDATE method in the rt_reg(4) registration file, and registered with scrgadm(1M).

The callback methods are passed prescribed operands and are expected to take certain actions to control the operation of the service on the cluster.

The paths to the callback method programs are declared in a resource type registration file, see rt_reg(4), by the resource type implementor. The cluster administrator uses scrgadm(1M) to register the resource type into the cluster configuration using the registration file. Also using scrgadm(1M), the registered resource type can then be used to create resources configured in resource groups managed by the RGM.

The RGM responds to events by automatically invoking the callback methods of the resources in the resource groups it manages. The callback methods are expected to take certain actions on the service represented by the resource, such as stopping or starting the service on a cluster node.

The exit value returned from the callback method indicates to the RGM whether the callback method succeeded or failed. The RGM either takes additional action in the event of a method failure, or records the failure in the resource state to indicate the need for administrative action.

OPERANDS

The following operands are supported:

-c — Operand for a VALIDATE method invocation. Indicates that the method is being called at the time of resource creation to validate the initial setting of all resource and resource group properties.

A VALIDATE invocation will either be passed a `-c` or `-u` flag, but not both.

The `-c` flag indicates that there will also be `-r` and `-x` operands passed giving values for all properties and extension properties in the resource, and `-g` operands passed giving values for all properties in the resource group.

`-g` *prop*`=val` — The operand provides the value of a resource group property to a VALIDATE method. The *prop* is the name of a resource group property, and `val` is the value of the property when the administrator creates the resource, or the value set when the resource group containing the resource is updated.

There might be several `-g` operands passed in a VALIDATE call.

`-G` *group* — The name of the resource group in which the resource is configured.

`-r` *prop*`=val` — The operand provides the value for a system-defined resource property to a VALIDATE method. The *prop* is the name of a system-defined resource property, and `val` the value set by the administrator on resource creation or update.

There might be several `-r` operands passed in a VALIDATE call.

`-R` *resource* — The name of the resource for which the method is invoked.

`-T` *type* — The name of the resource type of the resource.

`-u` — Operand for a VALIDATE method invocation. Indicates that the method is being called at the time of an administrative update of properties of an already existing resource, or update of the properties of the resource group containing the resource.

A VALIDATE invocation will either be passed a `-c` or `-u` flag, but not both.

The `-u` flag indicates that there will also be `-r`, `-x`, and `-g` passed giving values for all resource and resource group properties that were set by the administrative action. Only properties that have had values set in the update operation are passed. In contrast, the `-c` flag indicates that values for all properties are passed.

`-x` *prop*`=val` — The operand provides the value of a resource extension property to a VALIDATE method. The *prop* is the name of a resource extension property. An extension property is defined by the resource type implementation and declared in the paramtable of the resource type registration file. The `val` is the value set by the administrator on resource creation or update.

There might be several `-x` operands passed in a VALIDATE call.

USAGE

The callback methods are defined by the cluster RGM mechanism that invokes them. The methods are expected to execute operations on a cluster resource, and return an exit status reporting on the success of the operation. Following is a description of each callback method: how it is used by the RGM, what action it is expected to take, and the effect of a failure exit status.

START

The `START` method is invoked on a cluster node when the resource group containing the resource is brought online on that node. The administrator can toggle the state between on and off using the scswitch command. The `START` method activates the resource on a node.

`RGM` action on `START` method failure depends on the setting of the Failover_mode property of the resource. If Failover_mode is set to `SOFT` or `HARD`, the RGM will attempt to relocate the resource's group to another node, otherwise the RGM sets the resource's state to `START_FAILED`.

`STOP`

The `STOP` method is invoked on a cluster node when the resource group containing the resource is brought offline on that node. The administrator can toggle the state between on and off using the scswitch command. This method deactivates the resource if it is active.

`RGM` action on `STOP` method failure depends on the setting of the Failover_mode property of the resource. If Failover_mode is set to `HARD`, the `RGM` will attempt to forcibly stop the resource by aborting the node, otherwise the `RGM` sets the resource's state to `STOP_FAILED`.

`INIT`

The `INIT` method is invoked when the resource group containing the resource is put under the management of the RGM. It is called on nodes determined by the Init_nodes resource type property. The method is intended to do initialization of the resource.

`FINI`

The `FINI` method is invoked when the resource group containing the resource is removed from RGM management. It is called on nodes determined by the Init_nodes resource type property. The method is intended to do clean-up activities of the resource.

`BOOT`

The `BOOT` method is invoked when a node joins or rejoins the cluster as the result of being booted or rebooted. It is called on nodes determined by the Init_nodes resource type property. Similar to `INIT`, the method is intended to do initialization of the resource on nodes that join the cluster after the resource group containing the resource has already been brought online.

VALIDATE

The VALIDATE method is called when a resource is created, and also when administrative action updates the properties of the resource or its containing resource group. VALIDATE is called on the set of cluster nodes indicated by the Init_nodes property of the resource's type.

VALIDATE is called before the creation or update is applied, and a failure exit code from the method on any node causes the creation or update to be canceled.

When VALIDATE is called as the result of a resource being created, all system-defined, extension, and resource group properties are passed as parameters to VALIDATE. When VALIDATE is called as the result of an update to the resource, only the properties being updated are passed. You can use scha_resource_get and scha_resourcegroup_get to retrieve the properties of the resource not being updated.

If the VALIDATE method is implemented as a script, use logger(1) to write messages to the system log. If the VALIDATE method is implemented as a C program, use syslog(3C) to write messages to the system log.

UPDATE

The UPDATE method is called to notify a running resource that properties have been changed. UPDATE is invoked after an administration action succeeds in setting properties of a resource or its resource group. It is called on nodes where the resource is online. This method is intended to use the scha_resource_get and scha_resourcegroup_get access methods to read property values that can affect an active resource and adjust the running resource accordingly.

PRENET_START

An auxiliary to the START method, the PRENET_START method is intended to do start-up actions that are needed before the related network address is configured up. It is called on nodes where the START method is to be called. It is invoked after network addresses in the same resource group have been plumbed but before the addresses have been configured up and before the START method for the resource is called. The PRENET_START method is called before both the START method for the resource, and before the PRENET_START method of any other resource that depends on the resource.

PRENET_START failure has the same affect as START failure.

POSTNET_STOP

An auxiliary to the STOP method, the POSTNET_STOP method is intended to do shutdown actions that are needed after the related network address is configured down. It is called on nodes where the STOP method has been called. It is invoked after the network

addresses in the resource group have been configured down, and after the STOP method for the resource has been called, but before the network addresses have been unplumbed. The POSTNET_STOP method is called after both the STOP method for the resource and after the POSTNET_STOP method of any other resource that depends on the resource.

POSTNET_STOP failure has the same affect as STOP failure.

MONITOR_START — The MONITOR_START method is called after the resource is started, on the same node where the resource is started. It is intended to start a monitor for the resource. MONITOR_START may be called to restart monitoring that has been suspended.

MONITOR_START failure causes the RGM to set the resource state to MONITOR_FAILED.

MONITOR_STOP — The MONITOR_STOP method is called before the resource is stopped, on the same node where the resource is running. It is intended to stop a monitor for the resource. MONITOR_STOP may be called to suspend monitoring while the system disrupts global resources used by the resource. It is also called when monitoring is disabled by administrative action.

MONITOR_CHECK — The MONITOR_CHECK method is called before the resource group containing the resource is relocated to a new node as the result of a scha_control(3HA) or scha_control(1HA) request from a fault monitor. It may be called on any node that is a potential new master for the resource group. The MONITOR_CHECK method is intended to assess whether a node is healthy enough to run a resource. The MONITOR_CHECK method must be implemented in such a way that it does not conflict with the running of another method concurrently.

MONITOR_CHECK failure vetoes the relocation of the resource group to the node where the callback was invoked.

EXIT STATUS

0 — Successful completion. Communicates to the cluster RGM facility that the method succeeded.

non-0 — An error occurred.

The specific value of a failure exit status does not affect the RGM's action on failure. However, the exit status is recorded in the cluster log on method failure. A resource type implementation may define different non-0 exit codes to communicate error information to the administrator by way of the cluster log.

ENVIRONMENT VARIABLES

The Sun Cluster resource management callback methods are executed with root permission by the RGM cluster facility. The programs implementing the methods are expected to be installed with appropriate execution permissions, and for security, should not be writable.

Environment variables set for callback method execution are as follows:

```
HOME=/
PATH=/usr/bin:/usr/cluster/bin
LD_LIBRARY_PATH=/usr/cluster/lib
```

SIGNALS

If a callback method invocation exceeds its timeout period, the process is sent a SIGTERM signal. If the SIGTERM fails to stop the method execution, the process is sent SIGKILL.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`logger`(1), `scha_cmds`(1HA), `scrgadm`(1M), `syslog`(3C), `scha_calls`(3HA), `scha_control`(3HA), `rt_reg`(4), `signal`(3C), `attributes`(5)

NAME

scdsbuilder – Launch the GUI version of the Sun Cluster Data Service Builder

SYNOPSIS

```
scdsbuilder
```

DESCRIPTION

The `scdsbuilder` command launches the GUI version of the Sun Cluster Data Service Builder.

To run `scdsbuilder`, you must have a development version of Solaris 8 software or compatible versions, Java in your path, and JDK version 1.3.1 or compatible versions.

If a resource type developed with the Data Service Builder resides in the current directory, `scdsbuilder` automatically loads it and disables the `Create` button.

If the C compiler, `cc`(1B) is not in your path, then `scdsbuilder` disables the `C` option in the `C vs Ksh` question for the generated source code. If a resource type developed with the Data Service Builder and having its source code in C resides in the current directory, and the C compiler, `cc`, is not in your path, `scdsbuilder` returns with an error.

EXIT STATUS

The following exit values are returned:

`0`	Successful completion.
`>0`	An error occurred. The command did not complete.

FILES

install_directory`/rtconfig`	Contains information from the previous session; facilitates the tool's quit and restart feature.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`cc`(1B), `scdscreate(1HA)`, `scdsconfig(1HA)`, `attributes`(5)

NAME

scdsconfig – configure resource type template

SYNOPSIS

```
scdsconfig -s start_command [-u start_method_timeout] [-t stop_command]
     [-v stop_method_timeout] [-m probe_command] [-n probe_timeout]
     [-d working_directory]
```

DESCRIPTION

The `scdsconfig` command configures the resource type template that you previously created with `scdscreate(1HA)`. `scdsconfig` enables you to configure C-, Generic Data Service (GDS)-, or Korn shell-based templates for both network aware (client-server model) and non-network aware (clientless) applications.

`scdsconfig` configures application-specific commands to start, stop, and probe the application. You can also use `scdsconfig` to set timeout values for the `start`, `stop`, and `probe` commands. `scdsconfig` supports both network aware (client-server model) and non-network aware (client-less) applications. You can run `scdsconfig` from the same directory where `scdscreate` was run. You can also specify that same directory by using the `-d` option. `scdsconfig` configures the resource type template by placing the user-specified parameters at correct locations in the generated code. If C was the type of generated source code, this command also compiles the code. `scdsconfig` puts the output into a Solaris package that you can then install. This command creates the package in the `pkg` subdirectory under the $*vendor_id*$*resource_type_name* directory created by `scdscreate`.

OPTIONS

The following options are supported:

`-d` *working_directory*
: If `scdsconfig` is not run from the same directory where `scdscreate` was run, then this option is required to specify the directory where the resource type template was originally created.

`-m` *probe_command*
: This optional parameter specifies a command to periodically check the health of the network aware or non-network aware application. It must be a complete command line 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 successfully. An exit status other than `0` indicates that the application is failing to perform correctly. In this event, the resources of this resource type are either restarted on the same node or the resource group that contains the resource is failed over to another healthy node, depending on the failure history of the application in the past.

`-n` *probe_timeout*
: This optional parameter specifies the timeout, in seconds, for the probe command. The timeout must take into account system overloads to prevent false failures. The default value is 30 seconds.

`-s` *start_command*
: The start command starts the application. This command must be a complete command line that can

	be passed directly to a shell to start the application. You can include command line arguments to specify hostnames, port numbers, or other configuration data that is necessary to start the application. To create a resource type with multiple independent process trees, you specify a text file that contains the list of commands, one per line, to start the different process trees.
`-t` *stop_command*	This optional parameter specifies the stop command for the application. It must be a complete command line that can be passed directly to a shell to stop the application. If you omit this option, the generated code stops the application via signals. The stop command is allotted 80 percent of the timeout value to stop the application. If the stop command fails to stop the application within this period, a `SIGKILL` is allotted 15 percent of the timeout value to stop the application. If `SIGKILL` also fails to stop the application, the stop method returns with an error.
`-u` *start_method_timeout*	This optional parameter specifies the timeout, in seconds, for the start command. The timeout must take into account system overloads to prevent false failures. The default value is 300 seconds.
`-v` *stop_method_timeout*	This optional parameter specifies the timeout, in seconds, for the stop command. The timeout must take into account system overloads to prevent false failures. The default value is 300 seconds.

EXIT STATUS

The following exit values are returned:

`0`	The command completed successfully.
nonzero	An error occurred.

FILES

working_directory`/rtconfig`	Contains information from the previous session. Facilitates the tool's quit and restart feature.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO

`ksh`(1), `scdsbuilder`(1HA), `scdscreate`(1HA), `attributes`(5)

NAME

scdscreate – create a Sun Cluster resource type template

SYNOPSIS

```
scdscreate -V vendor_id -T resource_type_name -a [-s] [-n RT_version]
     [-d working_directory] [-k | -g]
```

DESCRIPTION

The `scdscreate` command creates a template for making an application highly available (HA) or scalable. This command enables you to create C-, Generic Data Service (GDS)-, or Korn shell-based templates for both network aware (client-server model) and non-network aware (clientless) applications.

You can create the template in one of two fundamentally different ways:

GDS — `scdscreate` creates a set of three driving scripts that work from a single resource type `SUNW.gds`, which is pre-installed on the cluster. These scripts are named start*RT_Name*, stop*RT_Name*, and remove*RT_Name* and starts, stops, and removes an instance of that application. In this model, the implementation of the `SUNW.gds` resource type that is pre-installed on the cluster is immutable.

Generated Source Code — `scdscreate` creates a template for a Sun Cluster resource type, whose instantiations run under the control of the Resource Group Manager (RGM) to make the given application highly available and scalable.

Either model can create templates for network aware (client-server model) and non-network aware (client-less) applications.

`scdscreate` creates a directory of the form $*vendor_id*$*resource_type_name* under *working_directory*. This directory contains the driving scripts, or the generated source, binary, and package files for the resource type. `scdscreate` also creates a configuration file, `rtconfig`, in which you can store configuration information for the resource type. `scdscreate` allows you to create only one resource type per directory. You must create different resource types in different directories.

OPTIONS

The following options are supported:

`-a` — This parameter specifies that the resource type that is being created is not network aware. `scdscreate` disables all the networking related code in the template that is created.

`-n` *RT_version* — This optional parameter specifies the version of the generated resource's type. If you omit this parameter, and you're creating a C- or Korn shell-based application, the text string `1.0` is used by default. If you omit this parameter, and you're creating a GDS-based application, the `RT_version` string of the

	GDS is used by default. The *RT_version* distinguishes between multiple registered versions, or upgrades, of the same base resource type. You cannot include the following characters in *RT_version*: blank, tab, slash (`/`), backslash (`\`), asterisk (`*`), question mark (`?`), comma (`,`), semicolon (`;`), left square bracket (`[`), or right square bracket (`]`).
`-d` *working_directory*	Creates the template for the resource type in a directory other than the current directory. If you omit this argument, `scdscreate` creates the template in the current directory.
`-g`	This optional parameter generates the GDS-based form of the template to make an application highly available or scalable.
`-k`	This optional parameter generates source code in Korn shell command syntax rather than in C. See `ksh`(1).
`-s`	This optional parameter indicates that the resource type is scalable. You can configure an instance (resource) of a scalable resource type into a failover resource group, and hence, turn off the scalability feature. If you omit this argument, `scdscreate` creates the template for a failover resource type.
`-T` *resource_type_name*	The resource type name and resource type version, in conjunction with the vendor ID, uniquely identifies the resource type that is being created.
`-V` *vendor_id*	The vendor ID is typically the stock symbol, or some other identifier of the vendor that is creating the resource type. `scdscreate` affixes the vendor ID, followed by a period (`.`) to the beginning of the resource type name. This syntax ensures that the resource type name remains unique if more than one vendor uses the same resource type name.

EXIT STATUS

`0`	The command completed successfully.
nonzero	An error occurred.

FILES

working_directory`/rtconfig`	Contains information from the previous session and facilitates the quit and restart feature of `scdscreate`.

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO `ksh`(1), `scdsbuilder`(1HA), `scdsconfig`(1HA), `attributes`(5), `rt_properties`(5)

Sun Cluster Data Services Developer's Guide for Solaris OS

NAME

scha_cluster_get – access cluster information

SYNOPSIS

`scha_cluster_get` `-O` *optag*...

DESCRIPTION

The `scha_cluster_get` command accesss information about a cluster. The command is intended to be used in shell script implementations of the callback methods for resource types that represent services controlled by the cluster's Resource Group Manager (RGM) facility. It provides the same information as the `scha_cluster_get(3HA)` function.

Information is output by the command to standard output in formatted strings as described in `scha_cmds(1HA)`. Output is takes the form of a string or strings on separate lines. The output can be stored in shell variables and parsed using shell facilities or `awk`(1) for use in scripts.

OPTIONS

The following options are supported:

`-O` *optag*

The *optag* argument indicates the information to be accessed. Depending on the *optag*, an additional argument may be needed to indicate the cluster node for which information is to be retrieved.

Note – *optag* options, such as `NODENAME_LOCAL` and `NODENAME_NODEID`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* options.

The following *optag* values are supported:

`NODENAME_LOCAL`
Outputs the name of the cluster node where command is executed.

`NODENAME_NODEID`
Outputs the name of the cluster node indicated by the numeric identifier. Requires an additional unflagged argument that is a numeric cluster node identifier.

`ALL_NODENAMES`
Outputs on successive lines the names of all nodes in the cluster.

`ALL_NODEIDS`
Outputs on successive lines the numeric node identifiers of all nodes in the cluster.

`NODEID_LOCAL`
Outputs the numeric node identifier for the node where the command is executed.

`NODEID_NODENAME`
Outputs the numeric node identifier of the node indicated by the name. Requires an additional unflagged argument that is the name of a cluster node.

`PRIVATELINK_HOSTNAME_LOCAL`
Outputs the hostname by which the node that the command is run on is addressed on the cluster interconnect.

`PRIVATELINK_HOSTNAME_NODE`
Outputs the hostname by which the named node is addressed on the cluster interconnect. Requires an additional unflagged argument that is the name of a cluster node.

`ALL_PRIVATELINK_HOSTNAMES`
Outputs on successive lines the hostnames by which all cluster nodes are addressed on the cluster interconnect.

`NODESTATE_LOCAL`
Outputs UP or DOWN depending on the state of the node where the command is executed.

`NODESTATE_NODE`
Outputs UP or DOWN depending on the state of the named node. Requires an additional unflagged argument that is the name of a cluster node.

`SYSLOG_FACILITY`
Outputs the number of the `syslog`(3C) facility that the RGM uses for log messages. The value is `24`, which corresponds to the `daemon` facility. You can use this value as the facility level in the `logger`(1) command to log messages in the cluster log.

`ALL_RESOURCEGROUPS`
Outputs on successive lines the names of all the resource groups that are being managed on the cluster.

`ALL_RESOURCETYPES`
Outputs on successive lines the names of all the resource types that are registered on the cluster.

`CLUSTERNAME`
Outputs the name of the cluster.

EXAMPLES

EXAMPLE 1 Using the `scha_cluster` Command in a Shell Script

The following shell script uses the `scha_cluster_get(1HA)` command to print whether each cluster node is up or down:

```
#!/bin/sh
nodenames=`scha_cluster_get -O All_Nodenames`
for node in $nodenames
do
     state=`scha_cluster_get -O NodeState_Node $node`
     printf "State of node: %s\n exit: %d\n value: %s\n" "$node" $? "$state"
done
```

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

non-0 An error occurred.

Failure error codes are described in scha_calls(3HA).

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Stable

SEE ALSO

awk(1), logger(1), sh(1), scha_cmds(1HA), scha_calls(3HA), scha_cluster_get(3HA), attributes(5)

NAME

scha_cmds – command standard output for `scha_cluster_get`, `scha_control`, `scha_resource_get`, `scha_resourcegroup_get`, `scha_resourcetype_get`, `scha_resource_setstatus`

SYNOPSIS

scha_command `-O` *optag...*

DESCRIPTION

The Sun Cluster `scha_cluster_get(1HA)`, `scha_control(1HA)`, `scha_resource_get(1HA)`, `scha_resourcegroup_get(1HA)`, `scha_resourcetype_get(1HA)`, and `scha_resource_setstatus(1HA)` commands are command-line implementations of the callback methods for resource types. See `rt_callbacks(1HA)`.

Resource types represent services that are controlled by the cluster's Resource Group Manager (RGM) facility. These commands provide a command line interface to the functionality of the `scha_calls(3HA)` C functions.

The `get` commands access cluster configuration information and all have the same general interface in that they take an `-O` *optag* operand that indicates the information to be accessed and output the results to standard output as formatted strings. Additional arguments might be needed depending on the command and *optag*. For information regarding the format for different *optag* results, see the Results Format section.

Note – *optag* options, for all `scha` commands, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* options.

The `scha_control(1HA)` command also takes an `-O` *optag* option that indicates a control operation, but does not produce output to standard output.

The `scha_resource_setstatus(1HA)` command sets the `STATUS` and `STATUS_MSG` properties of a resource that is managed by the RGM.

Result Formats

The format of strings that are output to the standard output by the commands depends on the type of the result that is indicated by the *optag* you include with the `-O` option. Formats for each type are specified in the following table. Format notation is described in `formats`(5).

Result Type	Format on Standard Output
`boolean`	`TRUE\n` or `FALSE\n`
`enum`	`%s\n`, the string name of an `enum` value

Result Type	Format on Standard Output
extension	%s\n, the type attribute of the extension property, which is one of the following values: STRING, INT, BOOLEAN, ENUM, or STRINGARRAY. Following the type information, the property value is output according to the formats for each type as follows: STRING as string, INT as int, BOOLEAN as boolean, ENUM as enum, STRINGARRAY as string_array
int	%d\n
status	%s\n%s\n, the first string is the status, which is one of the following enum values: DEGRADED, FAULTED, OFFLINE, ONLINE, or UNKNOWN. The second string is the status message.
string	%s\n
string_array	Each element in the array is output in the format %s\n. An asterisk, indicating all nodes or resources, can be returned for the GLOBAL_RESOURCES_USED and INSTALLED_NODES properties.
unsigned_int	%u\n
unsigned_int_array	Each element in the array is output in the format %u\n

optag Result Types

The following table specifies the valid *optag* values for different commands as well as the type of the result that is output according to the formats specified in the previous table.

optag Values for scha_cluster_get(1HA)	Result Type
ALL_NODEIDS	unsigned_int_array
ALL_NODENAMES	string_array
ALL_PRIVATELINK_HOSTNAMES	string_array
ALL_RESOURCEGROUPS	string_array
ALL_RESOURCETYPES	string_array
CLUSTERNAME	string
NODEID_LOCAL	unsigned_int
NODEID_NODENAME	unsigned_int
NODENAME_LOCAL	string

optag **Values for** `scha_cluster_get`**(1HA)**	**Result Type**
NODENAME_NODEID	string
NODESTATE_LOCAL	enum (UP, DOWN)
NODESTATE_NODE	enum (UP, DOWN)
PRIVATELINK_HOSTNAME_LOCAL	string
PRIVATELINK_HOSTNAME_NODE	string
SYSLOG_FACILITY	int

optag **Values for** `scha_control`**(1HA)**
CHECK_GIVEOVER
CHECK_RESTART
GIVEOVER
IGNORE_FAILED_START
RESOURCE_IS_RESTARTED
RESOURCE_RESTART
RESTART

optag **Values for** `scha_resource_get`**(1HA)**	**Result Type**
AFFINITY_TIMEOUT	int
ALL_EXTENSIONS	string_array
BOOT_TIMEOUT	int
CHEAP_PROBE_INTERVAL	int
EXTENSION	extension
FAILOVER_MODE	enum (NONE, HARD, SOFT, RESTART_ONLY, LOG_ONLY)
FINI_TIMEOUT	int
GROUP	string
INIT_TIMEOUT	int
LOAD_BALANCING_POLICY	string
LOAD_BALANCING_WEIGHTS	string_array
MONITORED_SWITCH	enum (DISABLED, ENABLED)

optag **Values for** `scha_resource_get`**(1HA)**	**Result Type**
MONITOR_CHECK_TIMEOUT	int
MONITOR_START_TIMEOUT	int
MONITOR_STOP_TIMEOUT	int
NETWORK_RESOURCES_USED	string_array
NUM_RESOURCE_RESTARTS	int
NUM_RG_RESTARTS	int
ON_OFF_SWITCH	enum (DISABLED, ENABLED)
PORT_LIST	string_array
POSTNET_STOP_TIMEOUT	int
PRENET_START_TIMEOUT	int
RESOURCE_DEPENDENCIES	string_array
RESOURCE_DEPENDENCIES_RESTART	string_array
RESOURCE_DEPENDENCIES_WEAK	string_array
RESOURCE_PROJECT_NAME	string
RESOURCE_STATE	enum (ONLINE, OFFLINE, START_FAILED, STOP_FAILED, MONITOR_FAILED, ONLINE_NOT_MONITORED, STARTING, STOPPING)
RESOURCE_STATE_NODE	enum (see RESOURCE_STATE for values)
RETRY_COUNT	int
RETRY_INTERVAL	int
R_DESCRIPTION	string
SCALABLE	boolean
START_TIMEOUT	int
STATUS	status
STATUS_NODE	status
STOP_TIMEOUT	int
THOROUGH_PROBE_INTERVAL	int
TYPE	string
TYPE_VERSION	string
UDP_AFFINITY	boolean

optag **Values for** `scha_resource_get`**(1HA)**	**Result Type**
UPDATE_TIMEOUT	int
VALIDATE_TIMEOUT	int
WEAK_AFFINITY	boolean

optag **Values for** `scha_resource_get`**(1HA) and** `scha_resourcetype_get`**(1HA)**	**Result Type**
API_VERSION	int
BOOT	string
FAILOVER	boolean
FINI	string
INIT	string
INIT_NODES	enum (RG_PRIMARIES, RT_INSTALLED_NODES)
INSTALLED_NODES	string_array. An asterisk (*) is returned to indicate all nodes.
IS_LOGICAL_HOSTNAME	boolean
IS_SHARED_ADDRESS	boolean
MONITOR_CHECK	string
MONITOR_START	string
MONITOR_STOP	string
PKGLIST	string_array
POSTNET_STOP	string
PRENET_START	string
RT_BASEDIR	string
RT_DESCRIPTION	string
RT_SYSTEM	boolean
RT_VERSION	string
SINGLE_INSTANCE	boolean
START	string
STOP	string

optag **Values for `scha_resource_get`(1HA) and `scha_resourcetype_get`(1HA)**	**Result Type**
UPDATE	string
VALIDATE	string

optag **Values for `scha_resourcegroup_get`(1HA)**	**Result Type**
AUTO_START_ON_NEW_CLUSTER	boolean
DESIRED_PRIMARIES	int
FAILBACK	boolean
GLOBAL_RESOURCES_USED	string_array (an asterisk (*) is returned to indicate all resources)
IMPLICIT_NETWORK_DEPENDENCIES	boolean
LOGICAL_HOST	boolean
MAXIMUM_PRIMARIES	int
NODELIST	string_array
PATHPREFIX	string
PINGPONG_INTERVAL	int
RESOURCE_LIST	string_array
RG_AFFINITIES	string_array
RG_DEPENDENCIES	string_array
RG_DESCRIPTION	string
RG_MODE	enum (FAILOVER, SCALABLE)
RG_PROJECT_NAME	string
RG_STATE	enum (UNMANAGED, ONLINE, OFFLINE, PENDING_ONLINE, PENDING_OFFLINE, ERROR_STOP_FAILED, ONLINE_FAULTED, PENDING_ONLINE_BLOCKED)
RG_STATE_NODE	enum (see RG_STATE for values)
RG_SYSTEM	boolean
RG_IS_FROZEN	boolean

EXIT STATUS

There is one set of exit values for all `scha` commands.

The exit values are the numeric values of the `scha_err_t` return codes of the corresponding C functions as described in `scha_calls`(3HA).

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Stable

SEE ALSO `awk`(1), `sh`(1), `rt_callbacks`(1HA), `scha_cluster_get`(1HA), `scha_control`(1HA), `scha_resource_get`(1HA), `scha_resourcegroup_get`(1HA), `scha_resourcetype_get`(1HA), `scha_resource_setstatus`(1HA), `scha_calls`(3HA), `attributes`(5), `formats`(5)

NAME

scha_control – request resource group control

SYNOPSIS

`scha_control` `-O` *optag* `-G` *group* `-R` *resource*

DESCRIPTION

The `scha_control` command requests the restart or relocation of a resource group that is under the control of the Resource Group Manager (RGM) cluster facility. This command is intended to be used in shell script implementations of resource monitors. It provides the same functionality as the `scha_control(3HA)` C function.

The exit code of the command indicates whether the requested action was rejected. If the request is accepted, the command does not return until the resource group or resource has completed going offline and back online. The fault monitor that called `scha_control(1HA)` might be stopped as a result of the group going offline and so might never receive the return status of a successful request.

You need `solaris.cluster.resource.admin` RBAC authorization to use this command. See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

OPTIONS

The following options are supported:

`-G` *group*
: Is the name of the resource group that is to be restarted or relocated. If the group is not online on the node where the request is made, the request is rejected.

`-O` *optag*
: Requests *optag* options.

  Note – *optag* options, such as `CHECK_GIVEOVER` and `CHECK_RESTART`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* options.

  The following *optag* values are supported:

  `CHECK_GIVEOVER`
  : Performs all the same validity checks that would be done for a `GIVEOVER` of the resource group named by the `-G` option, but does not actually relocate the resource group.

  `CHECK_RESTART`
  : Performs all the same validity checks that would be done for a `RESTART` of the resource group named by the `-G` option, but does not actually restart the resource group.

GIVEOVER
Requests that the resource group named by the -G option be brought offline on the local node, and online again on a different node of the RGM's choosing. Note that, if the resource group is currently online on two or more nodes and there are no additional available nodes on which to bring the resource group online, it can be taken offline on the local node without being brought online elsewhere. The request might be rejected depending on the result of various checks. For example, a node might be rejected as a host because the group was brought offline due to a GIVEOVER request on that node within the interval specified by the PINGPONG_INTERVAL property.

If the cluster administrator configures the RG_Affinities properties of one or more resource groups, and you issue a **scha_control GIVEOVER** request on one resource group, more than one resource group might be relocated as a result. The RG_Affinities property is described in rg_properties(5).

The MONITOR_CHECK method is called before the resource group that contains the resource is relocated to a new node as the result of a scha_control(3HA) or scha_control(1HA) request from a fault monitor.

The MONITOR_CHECK method may be called on any node that is a potential new master for the resource group. The MONITOR_CHECK method is intended to assess whether a node is healthy enough to run a resource. The MONITOR_CHECK method must be implemented in such a way that it does not conflict with the running of another method concurrently.

MONITOR_CHECK failure vetoes the relocation of the resource group to the node where the callback was invoked.

IGNORE_FAILED_START
Requests that, if the currently executing Prenet_start or Start method fails, the resource group is not to fail over, regardless of the setting of the Failover_mode property.

In other words, this optag value overrides the recovery action that is normally taken for a resource for which the Failover_Mode property is set to SOFT or HARD when that resource fails to start. Normally, the resource group fails over to a different node. Instead, the resource behaves as if Failover_Mode is set to NONE. The resource enters the START_FAILED state, and the resource group ends up in the ONLINE_FAULTED state, if no other errors occur.

This optag value is meaningful only when it is called from a Start or Prenet_start method that subsequently exits with a nonzero status or times out. This optag value is valid only for the current invocation of the Start or Prenet_start method. scha_control should be called with this optag value in a situation in which the Start method has determined that the resource cannot start successfully on another node. If this optag value is called by any other method, the error SCHA_ERR_INVAL is returned. This optag value prevents the "ping pong" failover of the resource group that would otherwise occur.

RESOURCE_IS_RESTARTED
: Requests that the resource restart counter for the resource named by the -R option be incremented on the local node, without actually restarting the resource.

  A resource monitor that restarts a resource directly without calling the RESOURCE_RESTART option of scha_control (for example, using pmfadm(1M)) can use this option to notify the RGM that the resource has been restarted. This incrementing is reflected in subsequent NUM_RESOURCE_RESTARTS queries of scha_resource_get(1HA).

  If the resource's type fails to declare the RETRY_INTERVAL standard property, the RESOURCE_IS_RESTARTED option of scha_control is not permitted, and scha_control returns exit 13 (SCHA_ERR_RT).

RESOURCE_RESTART
: Requests that the resource named by the -R option be brought offline and online again on the local node without stopping any other resources in the resource group. The resource is stopped and restarted by applying the following sequence of methods to it on the local node:

```
MONITOR_STOP
STOP
START
MONITOR_START
```

  If the resource's type does not declare a MONITOR_STOP and MONITOR_START method, then only the STOP and START methods are invoked to perform the restart. If the resource's type does not declare both a START and STOP method, scha_control fails with exit code 13 (SCHA_ERR_RT).

If a method invocation fails while restarting the resource, the RGM might set an error state, relocate the resource group, or reboot the node, depending on the setting of the FAILOVER_MODE property of the resource. For additional information, see the FAILOVER_MODE property in r_properties(5).

A resource monitor using this option to restart a resource can use the NUM_RESOURCE_RESTARTS query of scha_resource_get(1HA) to keep count of recent restart attempts.

The RESOURCE_RESTART function should be used with care by resource types that have PRENET_START, POSTNET_STOP, or both methods. Only the MONITOR_STOP, STOP, START, and MONITOR_START methods are applied to the resource. Network address resources on which this resource implicitly depends are not restarted and remain online.

RESTART

Requests that the resource group that is named by the -G option be brought offline, then online again, without forcing relocation to a different node. The request might ultimately result in relocating the resource group if a resource in the group fails to restart. A resource monitor using this option to restart a resource group can use the NUM_RG_RESTARTS query of scha_resource_get(1HA) to keep count of recent restart attempts.

The CHECK_GIVEOVER and CHECK_RESTART *optag* values are intended to be used by resource monitors that take direct action upon resources (for example, killing and restarting processes, or rebooting nodes) rather than invoking scha_control to perform a giveover or restart. If the check fails, the monitor should sleep for awhile and restart its probes rather than invoke its restart or failover actions. For more information, see scha_control(3HA).

-R *resource*

Is the name of a resource in the resource group, presumably the resource whose monitor is making the scha_control(1HA) request. If the named resource is not in the resource group, the request is rejected.

The setting of the Failover_mode property of the indicated resource might suppress the requested scha_control action. If Failover_mode is RESTART_ONLY, only scha_control RESOURCE_RESTART is permitted. Other requests, including GIVEOVER, CHECK_GIVEOVER, RESTART, and CHECK_RESTART, return the SCHA_ERR_CHECKS exit code and the requested giveover or restart action is not executed, producing only a

syslog message. If the Retry_count and Retry_interval properties are set on the resource, the number of resource restarts is limited to Retry_count attempts within the Retry_interval. If Failover_mode is LOG_ONLY, any scha_control request returns the SCHA_ERR_CHECKS exit code and the requested giveover or restart action is not executed, producing only a syslog message.

EXIT STATUS

The following exit values are returned:

0 The command completed successfully.

nonzero An error occurred.

Failure error codes are described in scha_calls(3HA).

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Stable

SEE ALSO

pmfadm(1M), rt_callbacks(1HA), scha_cmds(1HA), scha_resource_get(1HA), scha_control(3HA), scha_calls(3HA), attributes(5), r_properties(5), rg_properties(5)

NAME

scha_resource_get – access resource information

SYNOPSIS

scha_resource_get -O *optag* -R *resource* [-G *group*...]

DESCRIPTION

The scha_resource_get command accesses information about a resource that is under the control of the Resource Group Manager (RGM) cluster facility. You can use this command to query the properties of the resource's type, as described in rt_properties(5), as well as the properties of the resource, as described in r_properties(5).

scha_resource_get is intended to be used in shell script implementations of the callback methods for resource types that represent services controlled by the cluster's RGM. It provides the same information as the scha_resource_get(3HA) C function.

Information is output by the command to stdout in formatted strings as described in scha_cmds(1HA). Output is a string or several strings output on separate lines. The output can be stored in shell variables and parsed using shell facilities or awk(1) for further use by the script.

You need solaris.cluster.resource.read RBAC authorization to use this command. See rbac(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the pfsh(1), pfcsh(1), or pfksh(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run su(1M) to assume a role. You can also use pfexec(1) to issue privileged Sun Cluster commands.

OPTIONS

The following options are supported:

-G *group*
Is the name of the resource group in which the resource has been configured. Although this argument is optional, the command will run more efficiently if it is included.

-O *optag*
Indicates the information to be accessed. Depending on the *optag* that you specify, you might need to include an additional option to indicate the cluster node for which information is to be retrieved.

Note – *optag* options, such as AFFINITY_TIMEOUT and BOOT_TIMEOUT, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* options.

The following *optag* values retrieve the corresponding resource properties. The value of the named property of the resource is output. The RESOURCE_STATE, STATUS, NUM_RG_RESTARTS, and NUM_RESOURCE_RESTARTS properties refer to the value on the node where the command is executed (see r_properties(5)).

```
AFFINITY_TIMEOUT
BOOT_TIMEOUT
CHEAP_PROBE_INTERVAL
FAILOVER_MODE
FINI_TIMEOUT
INIT_TIMEOUT
LOAD_BALANCING_POLICY
LOAD_BALANCING_WEIGHTS
LOGICAL_HOSTNAMES_USED
MONITORED_SWITCH
MONITOR_CHECK_TIMEOUT
MONITOR_START_TIMEOUT
MONITOR_STOP_TIMEOUT
NETWORK_RESOURCES_USED
NUM_RESOURCE_RESTARTS
NUM_RG_RESTARTS
ON_OFF_SWITCH
PORT_LIST
POSTNET_STOP_TIMEOUT
PRENET_START_TIMEOUT
RESOURCE_DEPENDENCIES
RESOURCE_DEPENDENCIES_RESTART
RESOURCE_DEPENDENCIES_WEAK
RESOURCE_PROJECT_NAME
RESOURCE_STATE
RESOURCE_STATE_NODE
RETRY_COUNT
RETRY_INTERVAL
R_DESCRIPTION
SCALABLE
START_TIMEOUT
STATUS
STATUS_NODE
STOP_TIMEOUT
THOROUGH_PROBE_INTERVAL
TYPE
TYPE_VERSION
UDP_AFFINITY
UPDATE_TIMEOUT
VALIDATE_TIMEOUT
WEAK_AFFINITY
```

`STATUS_NODE`
Requires an unflagged argument that names a node. Outputs the value of the resource's `STATUS` property for the named node.

`RESOURCE_STATE_NODE`
Requires an unflagged argument that names a node. Outputs the value of the resource's `RESOURCE_STATE` property for the named node.

`EXTENSION`
Requires an unflagged argument that names an extension of the resource. Outputs the type of property followed by its value, on successive lines. Shell scripts might need to discard the type to obtain the value, as shown in EXAMPLES.

`ALL_EXTENSIONS`
Outputs on successive lines the names of all extension properties of the resource.

`GROUP`
Outputs the name of the resource group into which the resource is configured.

The following *optag* values retrieve the corresponding resource type properties. The value of the named property of the resource's type is output.

Note – *optag* options, such as `API_VERSION` and `BOOT`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* options.

For descriptions of resource type properties, see `rt_properties(5)`.

```
API_VERSION
BOOT
FAILOVER
FINI
INIT
INIT_NODES
INSTALLED_NODES
IS_LOGICAL_HOSTNAME
IS_SHARED_ADDRESS
MONITOR_CHECK
MONITOR_START
MONITOR_STOP
PKGLIST
POSTNET_STOP
PRENET_START
RT_BASEDIR
RT_DESCRIPTION
RT_SYSTEM
RT_VERSION
SINGLE_INSTANCE
START
STOP
UPDATE
VALIDATE
```

`-R` *resource*
Is the name of a resource that is being managed by the RGM cluster facility.

EXAMPLES

EXAMPLE 1 A Sample Script Using `scha_resource_get`

The following script is passed `-R` and `-G` arguments, which provide the required resource name and resource group name. Next, the `scha_resource_get` command accesses the `Retry_count` property of the resource and the `enum`-type `LogLevel` extension property of the resource.

```
#!/bin/sh

while getopts R:G: opt
do
     case $opt in
           R)      resource="$OPTARG";;
           G)      group="$OPTARG";;
     esac
done
```

EXAMPLE 1 A Sample Script Using `scha_resource_get` *(Continued)*

```
retry_count=`scha_resource_get -O Retry_count -R $resource \\
-G $group`
printf "retry count for resource %s is %d\n" $resource \\
$retry_count

LogLevel_info=`scha_resource_get -O Extension -R $resource \\
-G $group LogLevel`

# Get the enum value that follows the type information
# of the extension property.  Note that the preceding
# assignment has already changed the newlines separating
# the type and the value to spaces for parsing by awk.

loglevel=`echo $LogLevel_info | awk '{print $2}'`
```

EXIT STATUS

The following exit values are returned:

`0` The command completed successfully.

nonzero An error occurred.

Failure error codes are described in `scha_calls`(3HA).

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`awk`(1), `scha_cmds`(1HA), `scha_calls`(3HA), `scha_resource_get`(3HA), `attributes`(5), `r_properties`(5), `rt_properties`(5)

NAME

scha_resourcegroup_get – access resource group information

SYNOPSIS

scha_resourcegroup_get -O *optag* -G *group*...

DESCRIPTION

The scha_resourcegroup_get command accesses information about a resource group that is under the control of the Resource Group Manager (RGM) cluster facility.

This command is intended to be used in shell script implementations of the callback methods for resource types. These resource types represent services that are controlled by the cluster's RGM. This command provides the same information as the scha_resourcegroup_get(3HA) C function.

Information is output by the command to standard output in formatted strings as described in scha_cmds(1HA). Output is a string or several strings on separate lines. The output can be stored in shell variables and parsed using shell facilities or awk(1) for further use by the script.

You need solaris.cluster.resource.read RBAC authorization to use this command. See rbac(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the pfsh(1), pfcsh(1), or pfksh(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run su(1M) to assume a role. You can also use pfexec(1) to issue privileged Sun Cluster commands.

OPTIONS

The following options are supported:

-G *group* Is the name of the resource group.

-O *optag* Indicates the information that is to be accessed. Depending on the *optag* that you specify, you might need to include an additional operand to indicate the cluster node for which information is to be retrieved.

Note – *optag* options, such as DESIRED_PRIMARIES and FAILBACK, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* options.

The following *optags* retrieve the corresponding resource group properties. The value of the named property of the resource group is output. The RG_STATE property refers to the value on the node where the command is executed.

```
AUTO_START_ON_NEW_CLUSTER
DESIRED_PRIMARIES
FAILBACK
GLOBAL_RESOURCES_USED
IMPLICIT_NETWORK_DEPENDENCIES
MAXIMUM_PRIMARIES
NODELIST
PATHPREFIX
```

```
PINGPONG_INTERVAL
RESOURCE_LIST
RG_AFFINITIES
RG_DEPENDENCIES
RG_DESCRIPTION
RG_IS_FROZEN
RG_MODE
RG_PROJECT_NAME
RG_STATE
RG_STATE_NODE
RG_SYSTEM
```

Note – `RG_STATE_NODE` requires an unflagged argument that names a node. Outputs the value of the resource group's `RG_STATE` property for the named node.

EXAMPLES

EXAMPLE 1 A Sample Script Using `scha_resourcegroup_get`

The following script is passed a `-G` argument, which provides the required resource group name. Next, the `scha_resourcegroup_get` command is used to get the list of resources in the resource group.

```
#!/bin/sh

while getopts G: opt
do
     case $opt in
          G)      group="$OPTARG";;
     esac
done

resource_list=`scha_resourcegroup_get -O Resource_list -G $group`

for resource in $resource_list
do
     printf "Group: %s contains resource: %s\n" "$group" "$resource"
done
```

EXIT STATUS

The following exit values are returned:

`0` The command completed successfully.

nonzero An error occurred.

Failure error codes are described `scha_calls`(3HA).

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Stable

SEE ALSO awk(1), scha_cmds(1HA), scha_calls(3HA), scha_resourcegroup_get(3HA), attributes(5)

NAME

scha_resource_setstatus – command to set resource status

SYNOPSIS

scha_resource_setstatus -R *resource* -G *group* -s *status* [-m *msg*]

DESCRIPTION

The scha_resource_setstatus command sets the Status and Status_msg properties of a resource that is managed by the Resource Group Manager (RGM) cluster facility. This command is intended to be used by the resource's monitor to indicate the resource's state as perceived by the monitor. It provides the same functionality as the scha_resource_setstatus(3HA) C function.

A successful call to scha_resource_setstatus(1HA) causes the Status and Status_msg properties of the resource to be updated to the supplied values. The update of the resource status is logged in the cluster system log and is visible to cluster administration tools.

You need solaris.cluster.resource.admin RBAC authorization to use this command. See rbac(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the pfsh(1), pfcsh(1), or pfksh(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run su(1M) to assume a role. You can also use pfexec(1) to issue privileged Sun Cluster commands.

OPTIONS

The following options are supported:

-G *group*	Is the resource group that contains the resource.
-m *msg*	Is a string value. If no -m operand is given, the value of the resource's Status_msg is set to NULL.
-R *resource*	Names the resource whose status is to be set.
-s *status*	Is the value of *status*: OK, DEGRADED, FAULTED, UNKNOWN, or OFFLINE.

EXIT STATUS

The following exit values are returned:

0	The command completed successfully.
nonzero	An error occurred.
	Failure error codes are described in scha_calls(3HA).

EXAMPLES

EXAMPLE 1 Setting the Status of Resource R1

The following example sets the status of resource R1 in resource group RG2 to OK and sets the Status_msg to Resource R1 is OK:

```
scha_resource_setstatus -R R1 -G RG2 -s OK -m "Resource R1 is OK"
```

EXAMPLE 2 Setting the Status of Resource R1

The following example sets the status of `R1` in resource group `RG2` to `DEGRADED` and sets the `Status_msg` to `NULL`:

```
scha_resource_setstatus -R R1 -G RG2 -s DEGRADED
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Stable

SEE ALSO

`scha_cmds`(1HA), `scha_calls`(3HA), `scha_resource_setstatus`(3HA), `attributes`(5)

NAME

scha_resourcetype_get – access resource type information

SYNOPSIS

`scha_resourcetype_get` `-O` *optag* `-T` *type*

DESCRIPTION

The `scha_resourcetype_get` command accesses information about a resource type that is registered with the Resource Group Manager (RGM) cluster facility.

The command is intended to be used in shell script implementations of the callback methods for resource types that represent services controlled by the cluster's RGM. It provides the same information as the `scha_resourcetype_get(3HA)` C function.

Information is output by the command to `stdout` in formatted strings as described in `scha_cmds(1HA)`. Output is a string or several strings output on separate lines. The output might be stored in shell variables and parsed using shell facilities or `awk`(1) for further use by the script.

You need `solaris.cluster.resource.read` RBAC authorization to use this command. See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

OPTIONS

The following options are supported:

`-O` *optag* Indicates the information to be accessed.

Note – *optag* options, such as `API_VERSION` and `BOOT`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* options.

The following *optag* values retrieve the corresponding resource type properties. The value of the named property of the resource's type is output.

```
API_VERSION
BOOT
FAILOVER
FINI
INIT
INIT_NODES
INSTALLED_NODES
IS_LOGICAL_HOSTNAME
IS_SHARED_ADDRESS
MONITOR_CHECK
MONITOR_START
MONITOR_STOP
PKGLIST
POSTNET_STOP
```

```
PRENET_START
RT_BASEDIR
RT_DESCRIPTION
RT_SYSTEM
RT_VERSION
SINGLE_INSTANCE
START
STOP
UPDATE
VALIDATE
```

`-T` *type* Is the name of a resource type that is registered for use by the RGM cluster facility.

EXIT STATUS

The following exit values are returned:

`0` The command completed successfully.

nonzero An error occurred.

Failure error codes are described `scha_calls`(3HA).

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Stable

SEE ALSO

`awk`(1), `scha_cmds`(1HA), `scha_calls`(3HA), `scha_resourcetype_get`(3HA), `attributes`(5)

SC31 1m

NAME

cconsole , ctelnet, crlogin – multi window, multi machine, remote console, login and telnet commands

SYNOPSIS

`$CLUSTER_HOME/bin/cconsole` [*clustername*... | *hostname*...]

`$CLUSTER_HOME/bin/ctelnet` [*clustername*... | *hostname*...]

`$CLUSTER_HOME/bin/crlogin` [`-l` *user*] [*clustername*... | *hostname*...]

DESCRIPTION

These utilities initiate a multiple window connection to a set of specified hosts. There are three variations: one that is specifically intended for remote console access while the others provide remote logins using `rlogin`(1) or `telnet`(1).

Each utility starts a host window for each of the specified hosts, as well as a common window. Input directed into the common window is sent to each of these host windows.

This tool is useful for system administration tasks that require similar things to be done on each of several hosts. For tasks that are identical on all hosts, typing in the common window sends the characters to all of the hosts. However, the host windows are normal terminal windows so they can also be used one at a time (by moving the mouse into one of them and typing directly into it) to perform host specific tasks.

The common window also allows the user to select which hosts receive the characters typed in the common window, so only the specified hosts will receive input.

These utilities use entries in two different databases, `clusters(4)` and `serialports(4)`.

cconsole

Remote console access, using `cconsole` is provided through `telnet`(1). All normal `telnet` escape characters are available to the user. See `telnet`(1) for a complete listing of `telnet`(1) escape characters. Because there are a few `telnet` escapes that are commonly used, they are provided here as well. The escape character is Control-], specified below as ^].

^] quit	Quit the session. Analogous to `~.` in `tip`(1) and `rlogin`(1).
^] send brk	Send a break signal to the remote system. This is what is needed to halt the Sun CPU. The normal key board sequence is “L1-A.”

crologin

One of the options provided with `rlogin`(1) is also provided with the `crlogin` utility:

`-l` *user*	Specify a username, *user* for the remote login. The default is to use the local username. The argument value is remembered so hosts and clusters specified later can use the `-l` option when making the connection.

ctelnet

The `ctelnet` utility is similar to `cconsole` except the connection is directly over the Internet.

ENVIRONMENT VARIABLES

The following enviornment variables affect the execution of these utilities:

`CLUSTER_HOME` Location of Sun Cluster System tools. Defaults to `/opt/SUNWcluster`.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWccon
Interface Stability	Stable

SEE ALSO

`rlogin`(1), `telnet`(1), `tip`(1), `chosts`(1M), `cports`(1M), `clusters`(4), `serialports`(4), `attributes`(5)

NOTES

The standard set of X Window System command line arguments are accepted.

NAME

ccp – the Sun Cluster System Cluster Control Panel GUI

SYNOPSIS

`$CLUSTER_HOME/bin/ccp` [*clustername*]

DESCRIPTION

The `ccp` utility is a launch pad for the `cconsole`(1M), `ctelnet`(1M), and `crlogin`(1M) cluster utilities.

`ccp` also accepts the standard set of X Window System command line arguments.

OPERANDS

The following operands are supported:

clustername — If provided, this option could be passed on as an argument to a tool in ccp's set of tools. The *clustername* argument can be specified by adding `$CLUSTER` in a tool's command line property.

ENVIRONMENT VARIABLES

The following environment varaiables affect the exectution of the `ccp` utility:

`CLUSTER_HOME` — Location of cluster tools. Defaults to `/opt/SUNWcluster`.

`CCP_CONFIG_DIR` — Location of the tools' configuration files containing tool properties. Defaults to `/opt/SUNWcluster/etc/ccp`.

FILES

`$CLUSTER_HOME/etc/ccp/*`

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWccon
Interface Stability	Unstable

SEE ALSO

`cconsole`(1M), `ctelnet`(1M), `crlogin`(1M), `attributes`(5)

NAME

chosts – expand cluster names into host names

SYNOPSIS

`$CLUSTER_HOME/bin/chosts` *name* [*name*...]

DESCRIPTION

The `chosts` utility expands the arguments into a list of host names.

OPERANDS

The following operands are supported:

name The parameter *name* can be a hostname or a cluster name. If *name* is a hostname, it is expanded to be a hostname. If *name* is a cluster name, that is, an entry exists in the `/etc/clusters` database (or a NIS or NIS+ map), it is expanded into the list of hosts that make up that cluster, as specified in the database. The list is typically used by programs that wish to operate on a list of hosts.

If an entry for `clusters` has been made in the `/etc/nisswitch.conf` file, then the order of lookups is controlled by that entry. If there is no such file or no such entry, then the nameservice look up order is implicitly `nis files`.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWccon
Interface Stability	Unstable

SEE ALSO

`cconsole`(1M), `crlogin`(1M), `ctelnet`(1M), `cports`(1M), `clusters`(4), `attributes`(5)

NAME

cl_eventd – Cluster event daemon

SYNOPSIS

/usr/cluster/lib/sc/cl_eventd [-v]

DESCRIPTION

The cl_eventd daemon is started at boot time to monitor system events that are generated by other cluster components. This daemon also forwards these events to other cluster nodes. Only the events of class EC_Cluster are forwarded to other cluster nodes.

OPTIONS

The following option is supported:

-v Send additional troubleshooting and debugging information to syslogd(1M).

FILES

/usr/cluster/lib/sc/cl_eventd Cluster event daemon

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu

SEE ALSO

sysevent(1M), syslog(3C)

NOTES

The cl_eventd daemon does not provide a publicly accessible interface.

NAME

cports – expand host names into <host, server, port> triples

SYNOPSIS

$CLUSTER_HOME/bin/cports *hostname* [*hostname...*]

DESCRIPTION

The `cports` utility expands the *hostname* arguments into a list of `<host, server, port>` triples. The returned information is used to access the serial port consoles of the named hosts by way of the terminal server returned in the triples.

If an entry for `serialports` has been made in the `/etc/nisswitch.conf` file, then the order of lookups is controlled by that entry. If there is no such file or no such entry, then the nameservice look up order is implicitly `nis files`.

EXAMPLES

EXAMPLE 1 Using the `cports` Command

If the /etc/serialports file contains the entry:

```
pepsi    soda-tc    5002
```

this command:

```
% cports pepsi
```

prints the string:

```
pepsi soda-tc 5002
```

This information can be used by the `telnet`(1) command to remotely access `pepsi`'s console:

```
% telnet soda-tc 5002
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWccon
Interface Stability	Unstable

SEE ALSO

`cconsole`(1M), `crlogin`(1M), `ctelnet`(1M), `chosts`(1M), `telnet`(1), `serialports`(4), `attributes`(5)

NAME

cconsole , ctelnet, crlogin – multi window, multi machine, remote console, login and telnet commands

SYNOPSIS

`$CLUSTER_HOME/bin/cconsole` [*clustername*... | *hostname*...]

`$CLUSTER_HOME/bin/ctelnet` [*clustername*... | *hostname*...]

`$CLUSTER_HOME/bin/crlogin` [`-l` *user*] [*clustername*... | *hostname*...]

DESCRIPTION

These utilities initiate a multiple window connection to a set of specified hosts. There are three variations: one that is specifically intended for remote console access while the others provide remote logins using `rlogin`(1) or `telnet`(1).

Each utility starts a host window for each of the specified hosts, as well as a common window. Input directed into the common window is sent to each of these host windows.

This tool is useful for system administration tasks that require similar things to be done on each of several hosts. For tasks that are identical on all hosts, typing in the common window sends the characters to all of the hosts. However, the host windows are normal terminal windows so they can also be used one at a time (by moving the mouse into one of them and typing directly into it) to perform host specific tasks.

The common window also allows the user to select which hosts receive the characters typed in the common window, so only the specified hosts will receive input.

These utilities use entries in two different databases, `clusters(4)` and `serialports(4)`.

cconsole

Remote console access, using `cconsole` is provided through `telnet`(1). All normal `telnet` escape characters are available to the user. See `telnet`(1) for a complete listing of `telnet`(1) escape characters. Because there are a few `telnet` escapes that are commonly used, they are provided here as well. The escape character is Control-], specified below as ^].

^] quit	Quit the session. Analogous to `~.` in `tip`(1) and `rlogin`(1).
^] send brk	Send a break signal to the remote system. This is what is needed to halt the Sun CPU. The normal key board sequence is “L1-A.”

crologin

One of the options provided with `rlogin`(1) is also provided with the `crlogin` utility:

`-l` *user*	Specify a username, *user* for the remote login. The default is to use the local username. The argument value is remembered so hosts and clusters specified later can use the `-l` option when making the connection.

ctelnet

The `ctelnet` utility is similar to `cconsole` except the connection is directly over the Internet.

ENVIRONMENT VARIABLES

The following enviornment variables affect the execution of these utilities:

`CLUSTER_HOME` Location of Sun Cluster System tools. Defaults to `/opt/SUNWcluster`.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWccon
Interface Stability	Stable

SEE ALSO

`rlogin`(1), `telnet`(1), `tip`(1), `chosts`(1M), `cports`(1M), `clusters`(4), `serialports`(4), `attributes`(5)

NOTES

The standard set of X Window System command line arguments are accepted.

NAME

cconsole , ctelnet, crlogin – multi window, multi machine, remote console, login and telnet commands

SYNOPSIS

`$CLUSTER_HOME/bin/cconsole` [*clustername*... | *hostname*...]

`$CLUSTER_HOME/bin/ctelnet` [*clustername*... | *hostname*...]

`$CLUSTER_HOME/bin/crlogin` [`-l` *user*] [*clustername*... | *hostname*...]

DESCRIPTION

These utilities initiate a multiple window connection to a set of specified hosts. There are three variations: one that is specifically intended for remote console access while the others provide remote logins using `rlogin`(1) or `telnet`(1).

Each utility starts a host window for each of the specified hosts, as well as a common window. Input directed into the common window is sent to each of these host windows.

This tool is useful for system administration tasks that require similar things to be done on each of several hosts. For tasks that are identical on all hosts, typing in the common window sends the characters to all of the hosts. However, the host windows are normal terminal windows so they can also be used one at a time (by moving the mouse into one of them and typing directly into it) to perform host specific tasks.

The common window also allows the user to select which hosts receive the characters typed in the common window, so only the specified hosts will receive input.

These utilities use entries in two different databases, `clusters(4)` and `serialports(4)`.

cconsole

Remote console access, using `cconsole` is provided through `telnet`(1). All normal `telnet` escape characters are available to the user. See `telnet`(1) for a complete listing of `telnet`(1) escape characters. Because there are a few `telnet` escapes that are commonly used, they are provided here as well. The escape character is Control-], specified below as ^].

^] quit	Quit the session. Analogous to `~.` in `tip`(1) and `rlogin`(1).
^] send brk	Send a break signal to the remote system. This is what is needed to halt the Sun CPU. The normal key board sequence is “L1-A.”

crologin

One of the options provided with `rlogin`(1) is also provided with the `crlogin` utility:

`-l` *user*	Specify a username, *user* for the remote login. The default is to use the local username. The argument value is remembered so hosts and clusters specified later can use the `-l` option when making the connection.

ctelnet

The `ctelnet` utility is similar to `cconsole` except the connection is directly over the Internet.

ENVIRONMENT VARIABLES

The following enviornment variables affect the execution of these utilities:

`CLUSTER_HOME` Location of Sun Cluster System tools. Defaults to `/opt/SUNWcluster`.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWccon
Interface Stability	Stable

SEE ALSO

`rlogin`(1), `telnet`(1), `tip`(1), `chosts`(1M), `cports`(1M), `clusters`(4), `serialports`(4), `attributes`(5)

NOTES

The standard set of X Window System command line arguments are accepted.

NAME

halockrun – run a child program while holding a file lock

SYNOPSIS

/usr/cluster/bin/halockrun [-vsn] [-e *exitcode*] *lockfilename prog* [*args*]

DESCRIPTION

The halockrun utility provides a convenient means to claim a file lock on a file and run a program while holding that lock. As this utility supports script locking, this utiltiy is useful when programming in scripting languages such as the Bourne shell. See sh(1).

halockrun opens the file *lockfilename* and claims an exclusive mode file lock on the entire file. See fcntl(2) fcntl(2)). Then it runs the program *prog* with arguments *args* as a child process and waits for the child process to exit. When the child exits, halockrun releases the lock, and exits with the same exit code with which the child exited.

The overall effect is that the child *prog* is run as a critical section, and that this critical section is well-formed, in that no matter how the child terminates, the lock is released.

If the file *lockfilename* cannot be opened or created, then halockrun prints an error message on stderr and exits with exit code 99.

OPTIONS

The following options are supported:

e *exitcode*	Normally, errors detected by halockrun exit with exit code 99. The -e option provides a means to change this special exit code to a different value.
-n	The lock should be requested in non-blocking mode: if the lock cannot be granted immediately, halockrun exits immediately, with exit code 1, without running *prog*. This behavior is not affected by the -e option. Without the -n option, the lock is requested in blocking mode, thus, the halockrun utility blocks waiting for the lock to become available.
-s	Claim the file lock in shared mode, rather than in exclusive mode.
-v	Verbose output, on stderr.

EXIT STATUS

Errors detected by halockrun itself, such that the child process was never started, cause halockrun to exit with exit code 99. (This exit code value can be changed to a different value using the -e option. See OPTIONS.

Otherwise, halockrun exits with the same exit code with which the child exited.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO `fcntl`(2), `attributes`(5)

NAME

hatimerun – run child program under a timeout

SYNOPSIS

```
/usr/cluster/bin/hatimerun [-va] [-k signalname] [-e exitcode]
     -t timeOutSecs prog args
```

DESCRIPTION

The `hatimerun` utility provides a convenient facility for timing out the execution of another child, program. It is useful when programming in scripting languages, such as the Bourne shell. See `sh`(1).

The `hatimerun` utility runs the program *prog* with arguments *args* as a child subprocess under a timeout, and as its own process group. The timeout is specified in seconds, by the `-t` *timeOutSecs* option. If the timeout expires, then `hatimerun` kills the child subprocess's process group with a SIGKILL signal, and then exits with exit code 99.

OPTIONS

The following options are supported:

`-a` — Changes the meaning of `hatimerun` radically: instead of killing the child when the timeout expires, the `hatimerun` utility simply exits, with exit code 99, leaving the child to run asynchronously.

It is illegal to supply both the `-a` option and the `-k` option.

`-e` — Changes the exit code for the timeout case to some other value than 99.

`-k` — Specifies what signal is used to kill the child process group. The possible signal names are the same as those recognized by the `kill`(1) command. In particular, the signal name should be one of the symbolic names defined in the `<signal.h>` description. The signal name is recognized in a case-independent fashion, without the `SIG` prefix. It is also legal to supply a numeric argument to the `-k` option, in which case that signal number is used.

It is illegal to supply both the `-a` option and the `-k` option.

`-v` — Verbose output, on stderr.

EXIT STATUS

If the timeout occurs, then `hatimerun` exits with exit code 99 (which can be overridden to some other value using the `-e` option).

If the timeout does not occur but some other error is detected by the `hatimerun` utility (as opposed to the error being detected by the child program), then `hatimerunhatimerun` exits with exit code 98.

Otherwise, `hatimerun` exits with the child's exit status.

The `hatimerun` utility catches the signal SIGTERM. It responds to the signal by killing the child as if a timeout had occurred, and then exiting with exit code 98.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

`kill`(1), `sh`(1), `attributes`(5)

NAME

pmfadm – process monitor facility administration

SYNOPSIS

/usr/cluster/bin/pmfadm -c *nametag* [-a *action*] [[-e ENV_VAR=*env.var*...] | -E]

[-n *retries*] [-t *period*] [-C *level#*] *command* [*args-to-command...*]

/usr/cluster/bin/pmfadm -m *nametag* [-n *retries*] [-t *period*]

/usr/cluster/bin/pmfadm -s *nametag* [-w *timeout*] [*signal*]

/usr/cluster/bin/pmfadm -k *nametag* [-w *timeout*] [*signal*]

/usr/cluster/bin/pmfadm -l *nametag* [-h *host*]

/usr/cluster/bin/pmfadm -q *nametag* [-h *host*]

/usr/cluster/bin/pmfadm -L [-h *host*]

DESCRIPTION

The `pmfadm` utility provides the administrative, command-line interface to the process monitor facility.

The process monitor facility provides a means of monitoring processes, and their descendents, and restarting them if they fail to remain alive. The total number of failures allowed can be specified, and limited to a specific time period. After the maximum number of failures has occurred within the specified time period, a message is logged to the console, and the process is no longer restarted.

If an *action* program has been specified, it is called when the number of failures allowed has been reached. If the *action* program exits with non-zero status, the process nametag is removed from the process monitor facility. Otherwise, the process is restarted with the original parameters passed into `pmfadm`.

Processes that are started under control of the process monitor are run as the `uid` of the user that initiated the request. Only the original user, or root, can manipulate the nametag associated with those processes. Status information, however, is available to any caller, local or remote.

All spawned processes, and their descendent spawned processes, of the process that initially started are monitored. Only when the last process/sub-process exits does the process monitor attempt to restart the process.

OPTIONS

The following options are supported:

`-a` *action*
: The action program to be called when the process fails to stay alive. This program must be specified in a single argument to the `-a` flag, but can be a quoted string that contains multiple components. In either case, the string is executed as specified, with two additional arguments, the event that occurred (currently only `failed`), and the nametag associated with the process.

The current directory, and `PATH` environment variable, are reinstantiated before the command is executed. No other environment variables are, or should be assumed to be, preserved.

If the action program exits with status `0`, the process is started over again with the original arguments that were given to `pmfadm`. Any other exit status causes the nametag to cease to exist within the scope of the process monitor.

If no `-a` action is specified, the result is the same as if there were an action script specified which always exits non-zero.

`-c` *nametag*

Start a process, with *nametag* as an identifier. All arguments that follow the command-line flags are executed as the process of interest. The current directory, and PATH environment variable, are reinstantiated by the process monitor facility before the command is executed. No other environment variables are, or should be assumed to be, preserved.

If *nametag* already exists, `pmfadm` exits with exit status 1, with no side effects.

I/O redirection is not supported in the command line arguments. If this is necessary, a script should be created that performs this redirection, and used as the command that `pmfadm` executes.

`-C` *level#*

When starting a process, monitor it and its children up to and including level *level#*. The value of *level#* must be an integer greater than or equal to zero. The original process executed is at level `0`, its children are executed at level `1`, their children are executed at level `2`, and so on. Any new fork operation produces a new level of children.

This option provides more control over which processes get monitored. It is useful for monitoring servers that fork new processes.

When this option is not specified, all children are monitored, and the original process is not restarted until it and all its children have died.

	If a server forks new processes to handle client requests, it might be desirable to monitor only the server. The server needs to be restarted if it dies even if some client processes are still running. The appropriate monitoring level is `-C 0`. If, after forking a child, the parent exits, then it is the child that needs monitoring. The level to use to monitor the child is `-C 1`. When both processes die, the server is restarted.
`-e ENV_VAR=`*env.value*	An environment variable in the form `ENV_VAR=`*env.value* which is passed to the execution environment of the new process. This option can be repeated, so multiple environment variables can be passed. The default is not to use this option, in which case the `rpc.pmfd`(1M) environment plus the path of the `pmfadm` environment are passed.
`-E`	Pass the whole `pmfadm` environment to the new process. The default is not to use this option, in which case the `rpc.pmfd`(1M) environment plus the path of the `pmfadm` environment are passed. The `-e` and `-E` options are mutually exclusive, that is, both cannot be used in the same command.
`-h` *host*	The name of the host to contact. Defaults to localhost.
`-k` *nametag*	Send the specified signal to the processes associated with *nametag*, including any processes associated with the action program if it is currently running. The default signal, `SIGKILL`, is sent if none is specified. If the process and its descendants exit, and there are remaining retries available, the process monitor restarts the process. The signal specified is the same set of names recognized by the `kill`(1) command.
`-l` *nametag*	Print out status information about *nametag*. The output from this command is useful mainly for diagnostics and might be subject to change.
`-L`	Return a list of all tags running that belong to the user that issued the command, or if the user is root, all tags running on the server are shown.
`-m` *nametag*	Modify the number of retries, or time period over which to observe retries, for *nametag*. Once these parameters have been changed, the history of earlier failures is cleared.

`-n` *retries*	Number of retries allowed within the specified time period. The default value for this field is `0`, which means that the process is not restarted once it exits. The maximum value allowed is 100. A value of `-1` indicates that the number of retries is infinite.
`-q` *nametag*	Indicate whether *nametag* is registered and running under the process monitor. Returns `0` if it is, `1` if it is not. Other return values indicate an error.
`-s` *nametag*	Stop restarting the command associated with *nametag*. The signal, if specified, is sent to all processes, including the action script and its processes if they are currently executing. If a signal is not specified, none is sent. Stopping the monitoring of processes does not imply that they no longer exist. The processes remain running until they, and all of their descendents, have exited. The signal specified is the same set of names recognized by the `kill`(1) command.
`-t` *period*	Minutes over which to count failures. The default value for this flag is `-1`, which equates to infinity. If this parameter is specified, process failures that have occurred outside of the specified period are not counted.
`-w` *timeout*	When used in conjunction with the `-s` *nametag* or `-k` *nametag* flags, wait up to the specified number of seconds for the processes associated with *nametag* to exit. If the timeout expires, `pmfadm` exits with exit status `2`. The default value for this flag is `0`, meaning that the command returns immediately without waiting for any process to exit. If a value of `-1` is given, `pmfadm` waits indefinitely for the processes associated with the tag to exit. The `pmfadm` process does not release the RPC server thread that it uses until the RPC timeout period is reached. Therefore, avoid setting the `-w` *timeout* value to -1 unnecessarily.

EXAMPLES

EXAMPLE 1 Starting a Sleep Process That Will Not be Restarted

The following example starts a sleep process named `sleep.once` that will not be restarted once it exits:

```
example% pmfadm -c sleep.once /bin/sleep 5
```

EXAMPLE 2 Starting a Sleep Process and Restarting It

The following example starts a sleep process and restarts it, at most, one time:

```
example% pmfadm -c sleep.twice -n 1 /bin/sleep 5
```

EXAMPLE 3 Starting a Sleep Process and Restarting It

The following examples start a sleep process and restarts it, at most, twice per minute. It calls `/bin/true` when it fails to remain running beyond the acceptable number of failures:

```
example% pmfadm -c sleep.forever -n 2 -t 1 -a /bin/true /bin/sleep 60
```

EXAMPLE 4 Listing the Current Status of the `sleep.forever` Nametag

The following command lists the current status of the `sleep.forever` nametag:

```
example% pmfadm -l sleep.forever
```

EXAMPLE 5 Sending a SIGHUP to All Processes

The following command sends a `SIGHUP` to all processes associated with `sleep.forever`, waiting up to five seconds for all processes to exit.

```
example% pmfadm -w 5 -k sleep.forever HUP
```

EXAMPLE 6 Stopping the Monitoring of Processes and Sending a `SIGHUP`

The following command stops monitoring (restarting) processes associated with `sleep.forever`, and sends a `SIGHUP` to any processes related to it. This command returns as soon as the signals have been delivered, but possibly before all processes have exited.

```
example% pmfadm -s sleep.forever HUP
```

EXAMPLE 7 Listing All Tags Running That Belong to the User

If a user issues the following commands:

```
example% pmfadm -c sleep.once /bin/sleep 30
example% pmfadm -c sleep.twice /bin/sleep 60
example% pmfadm -c sleep.forever /bin/sleep 90
```

the output of the following command:

```
example% pmfadm -L
```

is

```
sleep.once sleep.twice sleep.forever
```

EXIT STATUS

The following exit values are returned:

`<0`	An error occurred.
`0`	Successful completion.
`1`	*nametag* doesn't exist, or there was an attempt to create a nametag that already exists.
`2`	The command timed out.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

`truss`(1), `rpc.pmfd`(1M), `attributes`(5)

NOTES

To avoid collisions with other controlling processes. `truss`(1) does not allow tracing a process that it detects as being controlled by another process by way of the `/proc` interface. Since `rpc.pmfd`(1M) uses the `/proc` interface to monitor processes and their descendents, those processes that are submitted to `rpc.pmfd` by way of `pmfadm` cannot be traced or debugged.

NAME

rpc.pmfd, pmfd – RPC-based process monitor server

SYNOPSIS

/usr/cluster/lib/sc/rpc.pmfd

/usr/cluster/lib/sc/pmfd

DESCRIPTION

`rpc.pmfd` is the Sun RPC server for serving the process monitor facility that is used by Sun Cluster software. This daemon initially starts when the system comes up.

`rpc.pmfd` must be started as superuser so commands that are queued to be monitored can be run as the user that submitted them.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

`truss`(1)`attributes`(5)

DIAGNOSTICS

Diagnostic messages are normally logged to the console.

NOTES

To avoid collisions with other controlling processes, `truss`(1) does not allow tracing a process that it detects as being controlled by another process by way of the `/proc` interface. As `rpc.pmfd` uses the `/proc` interface to monitor proceses and their descendents, those processes cannot be traced or debugged.

NAME

pnmd – Public Network Management (PNM) service daemon

SYNOPSIS

/usr/cluster/bin/pnmd [-d [-t [*tracefile*]]]

DESCRIPTION

pnmd is a server daemon for the Public Network Management (PNM) module. It is usually started up at system boot time. When it is started, it starts the PNM service.

in.mpathd(1M) does adapter testing and intra-node failover for all IP Network Multipathing (IPMP) groups in the local host.

pnmd keeps track of the local host's IPMP state and facilitates inter-node failover for all IPMP groups.

OPTIONS

The following options are supported:

-d	Display debug messages on stderr.
-t *tracefile*	When used with the -d option, it causes all debug messages to be redirected to *tracefile*. If *tracefile*. is omitted, /var/cluster/run/pnmd.log is used.

DIAGNOSTICS

pnmd is a daemon and has no direct stdin, stdout, or stderr connection to the outside. All diagnostic messages are logged through syslog(3C).

NOTES

pnmd must be run in super-user mode.

Due to the volume of debug messages generated, do not use the -t option for an extended period of time.

pnmd is started by the pnm startup script. It is started under the Process Monitoring Facility daemon pmfd. As such, if pnmd is killed by a signal, it is automatically restarted by pmfd.

The SIGTERM signal can be used to kill pnmd gracefully. Other signals should not be used to kill the daemon.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu
Interface Stability	Evolving

SEE ALSO

ifconfig(1M), in.mpathd(1M), syslog(3C), attributes(5)

NAME

rdt_setmtu – set the MTU size in RSMRDT driver

SYNOPSIS

/usr/cluster/bin/rdt_setmtu [MTU *size*]

DESCRIPTION

The `rdt_setmtu` command takes number of bytes as new MTU size and sets the global MTU size in RSMRDT driver. The RSMRDT driver uses the new MTU size for all the new instantiations of RSM connections. The existing RSM connections continue to use the old MTU size value. The MTU size should be a multiple of `64` (`0x40`) bytes otherwise `rdt_setmtu` does not set the MTU size in RSMRDT driver and returns an error. The `rdt_setmtu` when running without any argument, displays the MTU size of RSMRDT driver.

OPERANDS

The following operands are supported:

`MTU` *size* MTU size in bytes.

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

`1` An error occurred while setting MTU size.

This utility writes an error message to `stderr` when it exits with non-zero status.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes.

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscrdt
Interface Stability	Evolving

SEE ALSO

`attributes`(5)

NAME

rpc.pmfd, pmfd – RPC-based process monitor server

SYNOPSIS

/usr/cluster/lib/sc/rpc.pmfd

/usr/cluster/lib/sc/pmfd

DESCRIPTION

`rpc.pmfd` is the Sun RPC server for serving the process monitor facility that is used by Sun Cluster software. This daemon initially starts when the system comes up.

`rpc.pmfd` must be started as superuser so commands that are queued to be monitored can be run as the user that submitted them.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

`truss`(1)`attributes`(5)

DIAGNOSTICS

Diagnostic messages are normally logged to the console.

NOTES

To avoid collisions with other controlling processes, `truss`(1) does not allow tracing a process that it detects as being controlled by another process by way of the `/proc` interface. As `rpc.pmfd` uses the `/proc` interface to monitor proceses and their descendents, those processes cannot be traced or debugged.

NAME

sccheck – check for and report on vulnerable Sun Cluster configurations

SYNOPSIS

sccheck [-b] [-v *verbosity*] [-s *severity*] [-h *nodename*[,*nodename*]...] [-o *output-dir*]

sccheck [-b] [-v *verbosity*] [-W] [-h *nodename*[,*nodename*]...] [-o *output-dir*]

DESCRIPTION

The sccheck utility examines Sun Cluster nodes for known vulnerabilities and configuration problems, and it delivers reports that describe all failed checks, if any. The utility runs one of these two sets of checks, depending on the state of the node that issues the command:

- **Preinstallation checks –** When issued from a node that is not running as an active cluster member, the sccheck utility runs preinstallation checks on that node. These checks ensure that the node meets the minimum requirements to be successfully configured with Sun Cluster software.
- **Cluster configuration checks –** When issued from an active member of a running cluster, the sccheck utility runs configuration checks on the specified or default set of nodes. These checks ensure that the cluster meets the basic configuration required for a cluster to be functional. The sccheck utility produces the same results for this set of checks regardless of which cluster node issues the command.

The sccheck utility runs configuration checks and uses the explorer(1M) utility to gather system data for check processing. The sccheck utility first runs single-node checks on each *nodename* specified, then runs multiple-node checks on the specified or default set of nodes.

Each configuration check produces a set of reports that are saved in the specified or default output directory. For each specified *nodename*, the sccheck utility produces a report of any single-node checks that failed on that node. Then the node from which sccheck was run produces an additional report for the multiple-node checks. Each report contains a summary that shows the total number of checks executed and the number of failures, grouped by check severity level.

Each report is produced in both ordinary text and in XML. The DTD for the XML format is available in the /usr/cluster/lib/sccheck/checkresults.dtd file. The reports are produced in English only.

The sccheck utility is a client-server program in which the server is started when needed by the inetd daemon. Environment variables in the user's shell are not available to this server. Also, some environment variables, in particular those that specify the non-default locations of Java and Sun Explorer software, can be overridden by entries in the /etc/default/sccheck file. The ports used by the sccheck utility can also be overridden by entries in this file, as can the setting for required minimum available disk space. The server logs error messages to syslog and the console.

OPTIONS

The following options are supported:

-b
: Specifies a brief report. This report contains only the summary of the problem and the severity level. Analysis and recommendations are omitted.

You need `solaris.cluster.system.read` RBAC authorization to use this command option. See `rbac`(5).

`-h` *nodename*[,*nodename*]…
Specifies the nodes on which to run checks. If the `-h` option is not specified, the `sccheck` utility reports on all active cluster members.

This option is only legal when issued from an active cluster member.

`-o` *output-dir*
Specifies the directory in which to save reports. *output-dir* must already exist or be able to be created by the `sccheck` utility. Any previous reports in *output-dir* are overwritten by the new reports.

If the `-o` option is not specified, `/var/cluster/sccheck/reports.`*yyyy–mm–dd*`:`*hh*`:`*mm*`:`*ss* is used as *output-dir* by default, where *yyyy–mm–dd*:*hh*:*mm*:*ss* is the year-month-day:hour:minute:second when the directory was created.

`-s` *severity*
Specifies the minimum severity level to report on, where *severity* is a number in the range of 1 to 4 that indicates one of the following severity levels:

1. Low
2. Medium
3. High
4. Critical Each check has an assigned severity level. Specifying a severity level will exclude any failed checks of lesser severity levels from the report. When the `-s` option is not specified, the default severity level is 0, which means that failed checks of all severity levels are reported.

The `-s` option is mutually exclusive with the `-W` option.

`-v` *verbosity*
Specifies the `sccheck` utility's level of verbosity, where *verbosity* is a number in the range of 0 to 2 that indicates one of the following verbosity levels:

- 0: No progress messages. This level is the default.
- 1: Issues `sccheck` progress messages.
- 2: Issues Sun Explorer and more detailed `sccheck` progress messages.

You need `solaris.cluster.system.read` RBAC authorization to use this command option. See `rbac`(5).

The `-v` option has no effect on report contents.

`-W`
Disables any warnings. The report generated is equivalent to `-s3`.

The `-W` option is mutually exclusive with the `-s` option. The `-W` option is retained for compatibility with prior versions of the `sccheck` utility.

You need `solaris.cluster.system.read` RBAC authorization to use this command option. See `rbac`(5).

EXIT STATUS

The following exit values are returned:

`0` The command completed successfully. No violations were reported.

`1-4` The code indicates that the highest severity level of all violations was reported.

`100+` An error has occurred. Some reports might have been generated.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu, SUNWscsck
Interface Stability	Evolving

FILES

`/etc/default/sccheck`

`/usr/cluster/lib/sccheck/checkresults.dtd`

`/var/cluster/sccheck/reports.`*yyyy*–*mm*–*dd*`:`*hh*`:`*mm*`:`*ss*

SEE ALSO

`explorer`(1M), `sccheckd(1M)`, `scinstall(1M)`, `attributes`(5)

Sun Cluster Software Installation Guide, *Sun Cluster System Administration Guide*

NAME

sccheckd – service for the sccheck utility

SYNOPSIS

`sccheckd`

DESCRIPTION

The `sccheckd` service is the server side of the client-server utility `sccheck`(1M).

The `inetd`(1M) daemon starts the `sccheckd` service. The service reads the `/etc/default/sccheck` file at startup and during execution. The service logs diagnostics and error messages to `syslog` and the console. The `sccheckd` service has no direct connection to `stdin`, `stdout`, or `stderr`.

The `sccheckd` service exits when the last client connection exits.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu
Interface Stability	Evolving

FILES

`/etc/default/sccheck`

SEE ALSO

`sccheck`(1M)

NAME

scconf – update the Sun Cluster software configuration

SYNOPSIS

scconf -a [-Hv] [-h *node_options*] [-A *adapter_options*] [-B *junction_options*] [-m *cable_options*] [-P *privatehostname_options*] [-q *quorum_options*] [-D *devicegroup_options*] [-T *authentication_options*]

scconf -c [-Hv] [-C *cluster_options*] [-A *adapter_options*] [-B *junction_options*] [-m *cable_options*] [-P *privatehostname_options*] [-q *quorum_options*] [-D *devicegroup_options*] [-T *authentication_options*] [-w *heartbeat_options*]

scconf -r [-Hv] [-h *node_options*] [-A *adapter_options*] [-B *junction_options*] [-m *cable_options*] [-q *quorum_options*] [-D *devicegroup_options*] [-T *authentication_options*]

scconf -p [-Hv [v]]

scconf [-H]

DESCRIPTION

The scconf command manages the Sun Cluster software configuration. You can use scconf to add items to the configuration, to change properties of previously configured items, and to remove items from the configuration. In each of these three forms of the command, options are processed in the order in which they are typed on the command line. All updates associated with each option must complete successfully before the next option is considered.

The scconf command can only be run from an active cluster node. As long as the node is active in the cluster, it makes no difference which node is used to run the command. The results of running the command are always the same, regardless of the node used.

The -p option of scconf enables you to print a listing of the current configuration.

All forms of the scconf command accept the -H option. Specifying -H displays help information, and all other options are ignored and not executed. Help information is also printed when scconf is invoked without options.

OPTIONS

Basic Options

The following option is common to all forms of the scconf command:

-H
: If this option is specified on the command line at any position, prints help information. All other options are ignored and are not executed. Help information is also printed if scconf is invoked with no options.

The following options modify the basic form and function of the scconf command. None of these options can be combined on the same command line.

-a
: Specifies the add form of the scconf command. The -a option can be used to add or initialize most of the items that are used to define the software configuration of a Sun Cluster. Additional options are used with -a to specify elements (adapter, junction, or device group options, for example) and their associated properties to be

added. Any number of these additional options can be combined on the same command line, as long as they are for use with the `-a` option.

`-c`
Specifies the `change` form of the `scconf` command. The `-c` option is used to change properties of items already configured as part of the Sun Cluster software configuration. Additional options are used with `-c` to specify new or changed properties. Any number of these additional options can be combined on the same command line, as long as they are for use with the `-c` option.

`-p`
Specifies the `print` form of the `scconf` command. The `-p` option prints a listing of the current Sun Cluster configuration elements and their associated properties that you can configure with `scconf`. This option can be combined with one or more `-v` options to print more verbose listings.

`-r`
Specifies the `remove` form of the `scconf` command. The `-r` option is used to remove items from the Sun Cluster software configuration. Additional options are used with `-r` to specify the items to delete from the configuration. Any number of these additional options can be combined on the same command line, as long as they are for use with the `-r` option.

Additional Options

The following additional options can be combined with one or more of the previously described basic options. Refer to the SYNOPSIS section to see the options that can be used with each form of `scconf`.

The additional options are as follows:

`-A` *adapter_options*
Adds, removes, or changes the properties of a cluster transport adapter. The node on which the given adapter is hosted need not be active in the cluster for these operations to succeed. The `-A` *adapter_options* for each of the three forms of the command that accept `-A` are described here.

- Use this syntax to specify `-A` *adapter_options* for the `add` form of the command:

  `-A name=`*adaptername*`,node=`*node*`[,vlanid=`*vlanid*`][,state=`*state*`] \`
  `[,`*other_options*`]`
- Use this syntax to specify `-A` *adapter_options* for the `change` form of the command:

  `-A name=`*adaptername*`,node=`*node*`[,state=`*state*`] \`
  `[,`*other_options*`]`
- Use this syntax to specify `-A` *adapter_options* for the `remove` form of the command:

  `-A name=`*name*`,node=`*node*

The `-A` option supports the following suboptions:

`trtype=`*type*
Specifies the transport type. This suboption must be included when `-A` is used with the `add` form of the command.

An example of a transport *type* is `dlpi`. See `sctransp_dlpi`(7P).

`name=`*adaptername*
Specifies the name of an adapter on a particular node. This suboption must be included with each occurrence of the `-A` option.

adaptername is constructed from a *device name*, immediately followed by a *physical-unit* number (for example, `hme0`).

`node=`*node*
Specifies the name of an adapter on a particular node. A `node` suboption is required for each occurrence of the `-A` option.

The *node* can be given either as a node name or node ID.

[`vlanid=`*vlanid*]
Specifies the VLAN ID of the tagged-VLAN adapter.

`state=`*state*
Changes the state of the adapter. You can use this suboption with the `change` form of the command. The `state` can be set to either `enabled` or `disabled`.

When an adapter is added to the configuration, its state is always set to `disabled`. By default, adding a cable to any of the ports on an adapter changes the state of both the port and the adapter to `enabled`. See `-m` *cable_options*.

Disabling an adapter also has the effect of disabling all ports associated with that adapter. However, enabling an adapter does not result in the enabling of its ports. To enable an adapter port, you must enable the cable to which the port is connected.

[*other_options*]
If other options are available for a particular adapter type, they can be used with `-A` in the add and change forms of the command. Refer to the cluster transport adapter man pages (for example, `scconf_transp_adap_hme`(1M), `scconf_transp_adap_eri`(1M), and `scconf_transp_adap_sci`(1M)) for information about special options.

You need `solaris.cluster.transport.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-B` *junction_options*
Adds, removes, or changes the properties of a cluster transport junction.

Examples of such devices can include, but are not limited to, Ethernet hubs, other switches of various types, and rings.

The `-B` *junction_options* for each of the three forms of the command that accept `-B` are described here.

- Use this syntax to specify `-B` *junction_options* for the `add` form of the command:

  `-B type=`*type*`,name=`*name*`[,`*other_options*`]`

- Use this syntax to specify `-B` *junction_options* for the `change` form of the command:

  `-B name=`*name*`[,state=`*state*`][,`*other_options*`]`
- Use this syntax to specify `-B` *junction_options* for the `remove` form of the command:

  `-B name=`*name*

The `-B` option supports the following suboptions:

`type=`*type*
: Specifies a cluster transport junction type. This suboption must be included when `-B` is used with the `add` form of the command.

  Ethernet hubs and SCI switches are examples of cluster transport junctions of type `switch`. The man pages `scconf_transp_jct_dolphinswitch`(1M) and `scconf_transp_jct_etherswitch`(1M) contain more information.

`name=`*name*
: Specifies the name of a cluster transport junction. A `name` suboption must be included with each occurrence of the `-B` option.

  name can be up to 256 characters in length. It is made up of either letters or digits, with the first character being a letter. Each transport junction name must be unique across the namespace of the cluster.

`state=`*state*
: Changes the state of a cluster transport junction. This suboption can be used with a `-B change` command. `state` can be set to either `enabled` or `disabled`.

  When a junction is added to the configuration, its state is always set to `disabled`. By default, adding a cable to any of the ports on a junction changes the state of both the port and the junction to `enabled`. See `-m` *cable_options*.

  Disabling a junction also has the effect of disabling all ports associated with that junction. However, enabling a junction does not result in the enabling of its ports. To enable a junction port, you must enable the cable to which the port is connected.

[*other_options*]
: When other options are available for a particular junction type, they can be used with `-B` in the `add` and `change` forms of the command. Refer to the cluster transport junction man pages (for example, `scconf_transp_jct_dolphinswitch`(1M) and `scconf_transp_jct_etherswitch`(1M)) for information about special options.

You need `solaris.cluster.transport.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-C` *cluster_options*
: Changes the name of the cluster itself. This option can only be used with the `change` form of the command.

Specify *cluster_options* for the `change` form of the command as follows:

```
-C cluster=clustername cluster=clustername
```

This form of the command changes the name of the cluster to *clustername*.

`-D` *devicegroup_options*
Adds disk device groups to the configuration, changes or resets properties of existing device groups, or removes groups from the Sun Cluster device groups configuration. Other disk device group options (*other_options*) play a crucial role in adding or changing device groups and their options. Pay special attention to the man pages for the type-dependent disk device group options (for example, `scconf_dg_vxvm`(1M), `scconf_dg_sds`(1M), `scconf_dg_svm`(1M), and `scconf_dg_rawdisk`(1M)) when configuring any device group. Not all device group types support all three forms of the `-D` option. For example, `sds` device groups can normally only be used with the `change` form of the command to change certain attributes, such as the ordering of the node preference list.

The `add` form of the command can be used to either create device groups or to add nodes to existing device groups. For some device group types, the `add` form can also be used to add devices to a group. The `change` form of the command registers updates to change certain attributes associated with a group. The `remove` form of the command is used to either remove an entire device group or one or more of a group's components.

The `-D` *devicegroup_options* for each of the three forms of the `scconf` command that accept `-D` are as follows:

Add:

```
-D type=type,name=name,nodelist=node[:node]...
        [,preferenced={true | false}]
        [,numsecondaries=integer]
        [,failback={enabled | disabled}][,other_options]
```

Change:

```
-D name=name[,nodelist=node[:node]...]
        [,preferenced={true | false}]
        [,numsecondaries=integer]
        [,failback={enabled | disabled}][,other_options]
```

Remove:

```
-D name=name,nodelist=node[:node]...
```

The `-D` option supports the following suboptions:

`type=`*type*
Must be used with the `add` form of the command to indicate the type of disk device group to create (for example, `vxvm` or `rawdisk`).

`name=`*name*
Is the `name` of the disk device group and must be supplied with all three forms of the command.

`nodelist=`*node*[:*node*]…
Is a list of potential primary nodes that is required for some disk device group types when adding a group to the cluster. Refer to the man pages for the type-dependent disk device group for more information.

With the `add` form of the command, the `nodelist` is, by default, an ordered list indicating the preferred order in which nodes should attempt to take over as the primary node for a disk device group. However, if the `preferenced` suboption is set to `false` (see the next subsection), the first node to access a device in the group automatically becomes the primary node for that group. The `preferenced` suboption cannot be used when adding nodes to an existing device group. However, the `preferenced` suboption can be used when you create the group for the first time, or with the `change` form of the command.

To change the primary node order preference, you must specify the complete list of cluster nodes in the `nodelist` in the order that you prefer. You must also set the `preferenced` suboption to `true`.

When used with the `remove` form of the command, the `nodelist` suboption is used to remove the indicated nodes from the device group. Only by not providing a `nodelist` can the entire device group be removed. Simply removing all of the nodes from a device group does not necessarily remove that group.

[`preferenced={true | false}`]
Indicates the status of the preferred order of potential primary nodes for a disk device group. As long as the `preferenced` suboption is not set to `false`, node lists for newly created device groups indicate a preferred order in which nodes attempt to take over as the primary node for a disk device group.

If the `preferenced` suboption is not specified with an `add` that is used to create a device group, it is, by default, `false`. However, if the preferenced suboption is not specified with a `change`, it is, by default, set to `true` when `nodelist` is given.

The `preferenced` suboption cannot be used with an `add` that is used to add nodes to an established device group. In this case, the established node preference list setting is used.

[`numsecondaries=`*integer*]
Enables you to dynamically change the desired number of secondary nodes for a device group. A device group is an HA service that requires one node to act as a primary node and one or more nodes to act as secondary nodes. The secondary nodes of a device group are able to take over and act as the primary node if the current primary node fails.

This integer value should be greater than 0 but less than the total number of nodes in the specified group. The default is 1.

A system administrator can use the `numsecondaries` suboption to change the number of secondary nodes for a device group while maintaining a given level of availability. If a node in a device group is removed from the secondary nodes list, it is not able to take over and act as a primary node until it is converted back to a secondary node. Before making a change to the number of secondary nodes, you need to assess the impact on the secondary global file system.

The `numsecondaries` suboption only applies to nodes in a device group that are currently in cluster mode and can be used together with the node's `preferenced` suboption. If a device's `preferenced` suboption is enabled, the nodes that are least preferred are removed from the secondary nodes list first. If no node in a device group is flagged as preferred, the cluster randomly picks the node to remove.

When a device group's actual number of secondary nodes drops to less that the desired level due to node failures, nodes that were removed from the secondary nodes list are added back to the secondary list of nodes if they are currently in a cluster, belong to the device group, and are not currently a primary or a secondary node. The conversion starts with the node in the device group with the highest preference until the number of desired secondary nodes is matched.

If a node in the device group has a higher preference than an existing secondary node and joins the cluster, the node with the least preference is removed from the secondary nodes list and is replaced by the newly added node. This replacement only occurs when there are more actual secondary nodes than the desired level.

To set the desired number of secondary nodes to the system default (without having to know the default value), issue one of these commands:

```
# scconf -aD type=vxvm,name=foo, \
nodelist=node1:node2,numsecondaries=
```

or

```
# scconf -cD name=foo,numsecondaries=
```

The `numsecondaries` suboption can only be used with the `-a` option when a device group is created. The `numsecondaries` suboption cannot be used with the `-a` option to add a host to an existing device group.

[`failback={enabled | disabled}`]

Enables or disables the `failback` behavior of a disk device group with either the `add` or the `change` form of the command.

Specifies the behavior of the system should a disk device group primary node leave the cluster membership and later return.

When the node leaves the cluster membership, the disk device group fails over to the secondary node. When the failed node rejoins the cluster membership, the disk device group can either continue to be mastered by the secondary node, or fail back to the original primary node.

If `failback` is `enabled`, the disk device group becomes mastered by the original primary node. If `failback` is `disabled`, the disk device group continues to be mastered by the secondary node.

By default, `failback` is `disabled`.

[*other_options*]
You can use other disk device group type-dependent options with either the `add` or `change` form of the command. Refer to the appropriate man pages for more information (for example, `scconf_dg_vxvm`(1M), `scconf_dg_sds`(1M), `scconf_dg_svm`(1M), and `scconf_dg_rawdisk`(1M)).

You need `solaris.cluster.device.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-h` *node_options*
Adds or removes a node from the cluster configuration database. When used with the `add` form of `scconf`, both the new name and an internally generated node ID are added to the cluster configuration database. In addition, the new node is given a disk reservation key and a quorum vote count of zero. The name that is assigned to access the node over the cluster interconnect is initialized to `clusternode`*nodeid*`-priv`. See the `-p` option to learn more about printing configuration elements and their associated properties.

`scconf` cannot be used by itself to add a new node to the cluster. You can only use `scconf` to update the configuration database itself. `scconf` does not copy the configuration database onto the new node or create the necessary node identifier on the new node. To add a node to a cluster, use `scinstall`(1M).

When used with the `remove` form of `scconf`, all references to the node, including the last transport cable, all resource group references, and all device group references must be removed before `scconf` can be used to completely remove the node from the cluster configuration.

The node to be removed must not be configured for any quorum devices. In addition, you cannot remove a node from a three-node cluster unless there is at least one shared quorum device configured.

The system administration procedures in the Sun Cluster documentation describe how to remove a cluster node in more detail.

You must specify the `node=`*node* suboption with any occurrence of the `-h` option. For the `add` form of the command, the given *node* must be a node name.

Use this syntax to specify the `-h` *node_options* for the `add` form of the command:

`-h node=`*nodename*

For the `remove` form of the command, the *node* can be given either as a node name or node ID. Use this syntax to specify the `-h` *node_options* for the `remove` form of the command:

`-h node=`*node*

You need `solaris.cluster.node.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-m` *cable_options*

Helps to establish the cluster interconnect topology. This option helps by configuring the cables that are connecting the various ports that are found on the cluster transport adapters and junctions. Each new cable typically maps a connection either between two cluster transport adapters or between an adapter and a port on a transport junction. The `-m` *cable_options* for each of the forms of the command that accept `-m` are as follows:

- Use this syntax to specify the `-m` *cable_options* for the `add` form of the command:

  ```
  -m endpoint=[node:]name[@port],
      endpoint=[node:]name[@port][,noenable]
  ```

- Use this syntax to specify the `-m` *cable_options* for the `change` form of the command:

  ```
  -m endpoint=[node:]name[@port],state=state
  ```

- Use this syntax to specify the `-m` *cable_options* for the `remove` form of the command:

  ```
  -m endpoint=[node:]name[@port]
  ```

The `-m` option supports the following suboptions:

`endpoint=`[*node*:]*name*[*@port*]

Must be included with each occurrence of the `-m` option. For the `add` form of the command, two `endpoint` options must be specified. The *name* component of the option argument is used to specify the name of either a cluster transport adapter or cluster transport junction at one of the endpoints of a cable. If a *node* component is given, the *name* is the name of a cluster transport adapter. Otherwise, the *name* is the name of a cluster transport junction.

If a *port* component is not given, an attempt is made to assume a default port name. The default port for an adapter is always `0`. The default port name for a junction endpoint is equal to the node ID of the node attached to the other end of the cable. Refer to the cluster transport adapter and cluster transport junction man pages for more information about *port* assignments and other requirements (for example,`scconf_transp_adap_hme`(1M), `scconf_transp_adap_eri`(1M), `scconf_transp_adap_sci`(1M), `scconf_transp_jct_etherswitch`(1M), and `scconf_transp_jct_dolphinswitch`(1M)). Before a cable can be added, the adapters and junctions at each of the two endpoints of the cable must already be configured (see `-A` and `-B`).

`state=`*state*

Changes the state of a cable and the two endpoints to which it is connected. When a cable is enabled, the cable, its two ports, and the adapters or junctions associated with those two ports are all `enable`. However, when a cable is `disabled`, only the cable and its two ports are disabled. The state of the

adapters or junctions associated with the two ports remains unchanged. By default, the state of a cable, and its endpoints, is always set to `enabled` at the time that the cable is added to the configuration. But, to add a cable in the `disabled` state, use `noenable` as part of an add.

`noenable`
Can be used when adding a cable to the configuration. By default, when you add a cable, the state of the cable, the two ports to which it is connected, and the adapters or junctions on which the ports are found, are set to `enable`. But, if `noenable` is specified when you add a cable, the cable and its two endpoints are added in the disabled state. The state of the adapters or junctions on which the ports are found remains unchanged.

You need `solaris.cluster.transport.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-P` *privatehostname_options*
When used with either the `add` or `change` form of the command, specifies a host name alias to use for IP access of a given node over the private cluster interconnect, or transport. If not otherwise assigned, or if reset, the default private host name is `clusternode`*nodeid*`-priv`.

Private host names should never be stored in the `hosts`(4) database. A special nsswitch facility (see `nsswitch.conf`(4)) performs all host name lookups for private host names.

Both the `add` and `change` forms of `scconf` behave identically in relation to the `-P` option. The `-P` *privatehostname_options* for each of the two forms of the command that accept `-P` are as follows:

Add:

`-P node=`*node*`[,privatehostname=`*hostalias*`]`

Change:

`-P node=`*node*`[,privatehostname=`*hostalias*`]`

The `-P` option supports the following suboptions:

`node=`*node*
Provides the name or ID of the node to be assigned the private host name, or host alias, supplied with the `privatehostname` suboption.

[`privatehostname=`*hostalias*]
Supplies the host alias to be used for accessing a node over the private cluster interconnect, or transport. If no `privatehostname` suboption is specified, the private host name for the given `node` is reset to the default.

You need `solaris.cluster.transport.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-q` *quorum_options*

Manages shared cluster quorum devices and various cluster quorum properties. Pay special attention to the man pages for type-dependent quorum device options (for example, `scconf_quorum_dev_scsi`(1M) and `scconf_quorum_dev_netapp_nas`(1M).

The `add` and `remove` forms of the command add and remove shared quorum devices to or from the configuration. The `change` form of the command changes various cluster quorum configuration properties or states. The `-q` *quorum_options* available for each of the three forms of the command can be used to change the cluster quorum configuration as follows:

Add:

```
-q name=devicename,type={scsi | netapp_nas}

    For SCSI quorum devices only:
-q autoconfig[,noop]
```

Change:

```
-q node=node,{maintstate | reset}
-q name=devicename,{maintstate | reset}
-q reset
-q installmode


    For SCSI quorum devices only:
-q autoconfig[,noop]
```

Remove:

```
-q name=devicename
```

When `scconf` is interrupted or fails while performing quorum-related operations, quorum configuration information can become inconsistent in the cluster configuration database. If this occurs, either run the same `scconf` command again or run it with the `reset` suboption to reset the quorum information.

The `-q` option supports the following suboptions:

`autoconfig`

When used with the `add` form of the command, automatically chooses and assigns one quorum device in the two-node cluster. The quorum device is chosen from the available devices. If a quorum device is already configured, the command aborts.

When used with the `change` form of the command, automatically chooses and assigns one device that replaces all existing quorum devices in the two-node cluster. The quorum device is chosen from the available devices.

All available devices in the cluster must be qualified to be a quorum device. The `autoconfig` suboption does not assess whether an available device is qualified to be a quorum device.

If the cluster contains more than two nodes, the `autoconfig` suboption makes no changes to the quorum configuration. Do not use the `autoconfig` suboption if you intend to configure a NAS device as quorum.

`installmode`
Forces the cluster back into installation mode. While in `installmode`, nodes do not attempt to reset their quorum configurations at boot time. Also, while in this mode, many administrative functions are blocked. When a cluster is first installed, it is set up with `installmode` set. Once all of the nodes have joined the cluster for the first time, and shared quorum devices have been added to the configuration, issue `scconf -c -q reset` to reset the vote counts to their default values and to clear the `installmode` setting.

`name=`*devicename*
Specifies the name of an attached shared storage device to use when adding or removing a shared quorum device to or from the cluster. This suboption can also be used with the `change` form of the command to change the state of a quorum device.

Each quorum device must be connected, or ported, to at least two nodes in the cluster. It is not possible to use a non-shared disk as a quorum device.

The `change` form of `scconf` can be used with `-q name` to either put the device into a maintenance state or to reset the device's quorum configuration to the default. While in maintenance state, the device takes on a vote count of zero and, so, does not participate in forming quorum. When reset to the default, the vote count for the device is changed to N-1, where N is the number of nodes with nonzero vote counts that have ports to the device.

`node=`*node*
When used with the `add` form of the command, selects the nodes that should be configured with ports to the shared quorum device being added. This suboption can also be used with the `change` form of the command to change the quorum state of a node.

When the `node` suboption is used with the `change` form of the quorum update command, it is used to either place a node into maintenance state or to reset the node's quorum configuration to the default.

You must shut down a node before you can put it into maintenance state. `scconf` returns an error if you attempt to put a cluster member into maintenance state.

While in maintenance state, the node takes on a vote count of zero and, so, does not participate in quorum formation. In addition, any shared quorum devices configured with ports to the node have their vote counts adjusted down by one to reflect the new state of the node. When the node is reset to the default, its vote count is reset to 1 and the shared quorum device vote counts are re-adjusted back up. Unless the cluster is in `installmode`, the quorum configuration for each node is automatically reset at boot time.

A *node* can be specified as either a node name or a node ID.

{`maintstate`}
When used as a flag with the `change` form of the command, for either the `globaldev` or `node` suboptions, puts a shared quorum device or node into a quorum maintenance state. When in maintenance state, a shared device or node no longer participates in quorum formation. This feature can be useful when a node or device must be shut down for an extended period of maintenance. Once a node boots back into the cluster, under usual circumstances, it removes itself from maintenance mode.

It is not legal to specify both `maintstate` and `reset` with the same `-q` option.

{`reset`}
When used as a flag with the `change` form of the command, resets the configured quorum vote count of a shared quorum device or node. This option can be combined with either the `globaldev` or `node` suboptions, or it can be its own suboption.

If used by itself, the entire quorum configuration is reset to the default vote count settings. In addition, if `installmode` is set, it is cleared by a global quorum configuration reset. `installmode` cannot be reset on a two-node cluster unless at least one shared quorum device has been successfully configured.

[`,noop`]
Is valid with the `autoconfig` suboption. The command prints to standard output the list of quorum devices that the `autoconfig` suboption would add or change. The `autoconfig,noop` suboption makes no changes to the quorum configuration.

`type=`*`type`*
When used with the `add` form of the command, specifies the type of quorum device to create.

`scsi`
Specifies a shared disk quorum device. See `scconf_quorum_dev_scsi`(1M) for SCSI-type-specific options.

`netapp_nas`
Specifies a Network Appliance NAS quorum device. See `scconf_quorum_dev_netapp_nas`(1M) for NAS-type-specific options.

otheroptions
You can use other quorum-device-type-specific options. Refer to `scconf_quorum_dev_scsi`(1M) and `scconf_quorum_dev_netapp_nas`(1M) for details.

You need `solaris.cluster.quorum.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-T` *authentication_options*
Establishes authentication policies for nodes that are attempting to add themselves to the cluster configuration. Specifically, when a machine requests that it be added

to the cluster as a cluster node (see scinstall(1M)), a check is made to determine whether or not the node has permission to join. If the node has permission, the joining node is authenticated. By default, any machine is allowed to add itself to the cluster.

The `-T` *authentication_options* for each of the three forms of the command that accept `-T` are as follows:

Add:

`-T node=`*nodename*`[,...][,authtype=`*authtype*`]`

Change:

`-T authtype=`*authtype*

Remove:

`-T {node=`*nodename*`[,...] | all}`

The `-T` option supports the following suboptions:

`node=`*nodename*
: Adds or removes host names from the list of nodes that are able to install and configure themselves as nodes in the cluster. At least one `node` suboption is required for the `add` form of the command and is optional for `remove`. If the authentication list is empty, any host can request that it be added to the cluster configuration. However, if the list has at least one name in it, all such requests are authenticated using the authentication list.

  Illegal *nodename*s are accepted, including the node name of dot (`.`). The dot character is special in that if a *nodename* of `.` is added to the authentication list, all other names are removed. This feature prevents a host from attempting to install and configure itself in the cluster.

`all`
: You can clear the list of all node names by specifying `scconf -r -T all`. A cleared authentication list means that any node can attempt to install and configure itself in the cluster.

`authtype=`*authtype*
: Is used with either the `add` or `change` form of the command.

  The only currently supported authentication types (`authtype`) are `des` and `sys` (or `unix`). The default authentication type is `sys`, which provides the least amount of secure authentication.

  When `des`, or Diffie-Hellman, authentication is used, entries should be added to the `publickey`(4) database for each cluster node to be added before actually running scinstall(1M) to add the node.

You need `solaris.cluster.node.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-v`
When used with the `-p` option, requests a more verbose, or detailed, listing of the cluster configuration. If used with other options, additional information might be printed when an error is encountered.

You need `solaris.cluster.device.read`, `solaris.cluster.transport.read`, `solaris.cluster.resource.read`, `solaris.cluster.node.read`, `solaris.cluster.quorum.read`, and `solaris.cluster.system.read` RBAC authorizations to use this command option with `-p`. See `rbac`(5).

`-w` *heartbeat_options*
Changes the global heartbeat parameters of a cluster, which effectively changes the heartbeat parameters across all the adapters of the cluster.

Sun Cluster relies on heartbeats over the private interconnect to detect communication failures among cluster nodes. Reducing the heartbeat timeout enables Sun Cluster to detect failures more quickly, as the time that is required to detect failures decreases when you decrease the values of heartbeat timeout. Thus, Sun Cluster recovers more quickly from failures, consequently increasing the availability of your cluster.

The `-w` option supports the following suboptions:

`heartbeat_quantum=`*quantum_milliseconds*
Defines how often to send heartbeats. Sun Cluster uses a 1 second (1,000 milliseconds) heartbeat quantum by default. Specify a value between 100 milliseconds and 10,000 milliseconds.

`heartbeat_timeout=`*timeout_milliseconds*
The time interval after which, if no heartbeats are received from the peer nodes, the corresponding path is declared as down. Sun Cluster uses a 10 second (10,000 millisecond) heartbeat timeout by default. Specify a value between 2,500 milliseconds and 60,000 milliseconds.

Note – Even under ideal conditions, when you reduce the values of heartbeat parameters with `-w`, there is always a risk that spurious path timeouts and node panics might occur. Always test and thoroughly qualify the lower values of heartbeat parameters under relevant workload conditions before actually implementing them in your cluster.

USAGE

With the `-w` option, you can change only one heartbeat suboption at a time. When decreasing the values of heartbeat parameters, change `heartbeat_quantum` first, followed by `heartbeat_timeout`. When increasing the values of heartbeat parameters, change `heartbeat_timeout` first, followed by `heartbeat_quantum`.

Note – The value you specify for `heartbeat_timeout` must always be greater than or equal to five times the value you specify for `heartbeat_quantum` (`heartbeat_timeout` >= (5*`heartbeat_quantum`)).

You need `solaris.cluster.system.modify` RBAC authorization to use `-w`. See `rbac`(5).

If you change heartbeat parameters with `-w`, and you later choose to back out and go back to a previous version of Sun Cluster that does not support `-w`, you must first reset all heartbeat parameters to the original, default values that are supported in the previous version.

EXAMPLES

EXAMPLE 1 Decreasing the Heartbeat

The following example shows how to decrease the heartbeat quantum to 100 milliseconds from the Sun Cluster default of 1,000 milliseconds. This example also shows how to decrease the heartbeat timeout to 2500 milliseconds from the Sun Cluster default of 10,000 milliseconds.

```
phys-schost-1# scconf -c -w heartbeat_quantum=100
phys-schost-1# scconf -c -w heartbeat_timeout=2500
```

Because `heartbeat_timeout` must always be greater than or equal to five times `heartbeat_quantum`, you need to set `heartbeat_quantum` first. Otherwise, the requirement is not met. In other words, if `heartbeat_quantum` is currently set to the default 1,000 milliseconds, and if you were to set `heartbeat_timeout` to 2500 milliseconds, `heartbeat_timeout` would be *less* than five times `heartbeat_quantum`. The `scconf` command would consequently fail.

Once `heartbeat_quantum` is set to the correct value however, the requirement is maintained, and you can then set `heartbeat_timeout` to the decreased value.

EXAMPLE 2 Increasing the Heartbeat

The following example shows how to increase the heartbeat timeout and heartbeat quantum to Sun Cluster default values from the values to which you set these parameters in the previous example.

```
phys-schost-1# scconf -c -w heartbeat_timeout=10000
phys-schost-1# scconf -c -w heartbeat_quantum=1000
```

You set `heartbeat_timeout` first to maintain the requirement that `heartbeat_timeout` always be greater than or equal to five times `heartbeat_quantum`. Once `heartbeat_timeout` is set to the value you want, you can then set `heartbeat_quantum` to the new, increased value.

EXAMPLE 3 Typical Postinstallation Setup Operations

The following commands provide an example of a typical set of postinstallation setup operations that might be performed on a new two-node cluster. These commands add a shared quorum device to the cluster, clear `installmode`, configure a second set of cluster transport connections, and secure the cluster against other machines that might attempt to add themselves to the cluster:

```
phys-red# scconf -a -q globaldev=d0
phys-red# scconf -c -q reset
phys-red# scconf -a \
    -A trtype=dlpi,name=hme1,node=phys-red \
```

EXAMPLE 3 Typical Postinstallation Setup Operations *(Continued)*

```
    -A trtype=dlpi,name=hme1,node=phys-green \
    -m endpoint=phys-red:hme1,endpoint=phys-green:hme1
phys-red# scconf -a -T node=.
```

EXIT STATUS

The following exit values are returned:

`0`	The command completed successfully.
nonzero	An error has occurred.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscu`
Interface Stability	Evolving

SEE ALSO

`scconf_dg_rawdisk`(1M), `scconf_dg_sds`(1M), `scconf_dg_svm`(1M), `scconf_dg_vxvm`(1M), `scconf_quorum_dev_scsi`(1M), `scconf_quorum_dev_netapp_nas`(1M), `scconf_transp_adap_bge`(1M), `scconf_transp_adap_ce`(1M), `scconf_transp_adap_e1000g`(1M), `scconf_transp_adap_eri`(1M), `scconf_transp_adap_ge`(1M), `scconf_transp_adap_hme`(1M), `scconf_transp_adap_ibd`(1M), `scconf_transp_adap_qfe`(1M), `scconf_transp_adap_sci`(1M), `scconf_transp_adap_wrsm`(1M), `scconf_transp_jct_dolphinswitch`(1M), `scconf_transp_jct_etherswitch`(1M), `scconf_transp_jct_ibswitch`(1M), `hosts`(4), `nsswitch.conf`(4), `publickey`(4), `attributes`(5), `sctransp_dlpi`(7P)

WARNINGS

Use the `-w` option only when *all* nodes in a cluster are up. Do not use `-w` when any node in a cluster is down. Nodes might hang or panic as a result.

Clusters that contain one or more single-CPU nodes, or that contain more than eight nodes, are more likely to experience timeouts and node panics when the clusters run with low heartbeat parameter values.

Note – Even under ideal conditions, when you reduce the values of heartbeat parameters with `-w`, there is always a risk that spurious path timeouts and node panics might occur. Always test and thoroughly qualify the lower values of heartbeat parameters under relevant workload conditions before actually implementing them in your cluster.

NOTES

You should either back up the root file system on every node after changing the configuration with `scconf`, or keep a log of all changes. If you need to recover configuration changes between normal system backups, use the log to return to the most recent configuration.

Option lists specified with the `scconf` command are always executed in the order that you specify them on the command line. But, whenever possible, certain transport options (`-A`, `-B`, and `-m`) are processed by `scconf` as a single transaction against the cluster configuration database. Try to group all related options of this type together on a single command line to reduce overhead to the cluster.

The `-w` option works only in 3.1 8/04 and later versions of Sun Cluster that run on Solaris 8 Update 7 and later versions of Solaris.

NAME

scconf_dg_rawdisk – add, change or update rawdisk device group configuration

SYNOPSIS

```
scconf -a -D type=rawdisk, [generic_options]
     [,globaldev=gdev1,globaldev=gdev1,…] [,localonly=true]

scconf -a -D type=rawdisk, [generic_options]
     [,globaldev=gdev1,globaldev=gdev1,…]
     [,localonly=true | false]

scconf -c -D name=diskgroup,autogen=true

scconf -r -D device_service_name [,nodelist=node[:node]…]
     [,globaldev=gdev1,…]
```

DESCRIPTION

The `scconf_dg_rawdisk` utility adds, changes or updates rawdisk device group configuration

A rawdisk is a disk that is not being used as part of a volume manager volume or metadevice. Rawdisk device groups allow you to define a set of disks within a disk device group. At system boot, by default, a rawdisk device group is created for every Disk ID pseudo driver (DID) device in the configuration. By convention, the rawdisk device group names are assigned at initialization and are derived from the DID names. For every node added to a rawdisk disk device group, the `scconf` utility verifies that every device in the group is physically ported to the node.

The `scconf add` (`-a`) command can be used to create a rawdisk device group with multiple disk devices configured in it. A rawdisk device group is created for every disk device in the cluster at boot time. Before you can add a new rawdisk device group, devices to be used in the new group must be removed from the device group created at boot time. Then a new rawdisk device group can be created containing these devices. This is accomplished by creating a list of these devices in the `globaldev` option of `scconf` along with a potential primary node preference list in the `nodelist` option. If the device group already exists, only new nodes and global devices will be added and nodes or devices which are part of an existing device group will be ignored. If the `preferenced` suboption is not given at all with an `add` to create a new device group, then it is, by default, `false`. However, if the `preferenced` suboption is specified for the existing device group with a value of `true` or `false`, an error is returned. This is done in order to maintain the existing nodelist preference state. If a device group should be mastered by only a particular node then it should be configured with the `otheroption` set to `localonly=true`. Only one node can be specified in the nodelist to create a `localonly` device group.

The `scconf change` (`-c`) command is used to change the order of the potential primary node preference, to enable or disable failback, to set the desired nuber of secondarie, and to add more global devices to the device group.

If you want to change the order of node preference list, then all the nodes currently existing in the device group must be specified in the `nodelist`. In addition, if you are changing the the order of node preference, you must also set the `preferenced` suboption to `true`.

If the `preferenced` suboption is not specified with the change, the already established `true` or `false` setting is used.

New nodes cannot be added using the `change` form of the command. Change option can also be used for changing a device group to `localonly` device group and vice-versa. To change a device group to a `localonly` device group, set `otheroption` to `localonly=true`. Specify `localonly=false` to set it back to not the `localonly` device group. `nodelist` must already be set to a list of one node, or an error results. It is legal to specify a `nodelist` with the `change` form of the command, when you set `localonly` to `true`. This is, however, redundant, since the list can only contain the single node that is already configured. It would be an error to specify any other than the node that is already configured.

The `scconf remove` (`-r`) command can be used to remove the nodes, global devices, and the device group name from the cluster device group configuration. If nodes or global devices are specified with the device group name, they are removed from the device group first. After the last device and node are removed from the device group, the device group is also removed from cluster configuration. If only the name of the device group is given (no nodes or devices at all), the entire device group is removed.

If a rawdisk device name is registered in a rawdisk device group then it cannot be registered in an Solstice DiskSuite device group or a VERITAS Volume Manager device group.

OPTIONS

See `scconf(1M)` for the list of supported generic options.

The following action options are used to describe the actions performed by the command. Only one action option is allowed in the command.

The following action options are supported:

`-a` Add a new rawdisk device group to the cluster configuration. You can also use this option to change the device group configuration.

`-c` Change the ordering of the node preference list, change preference and failback policy, change the desired number of secondaries, and also add more devices to the device group with the `globaldev` option. It is also used to set a device group as local only.

`-r` Remove the rawdisk device group name from the cluster.

The `autogen` flag is an indicator of the `scstat` and `scconf` commands. These two commands do not list devices with the `autogen` property unless the `-v` command line option is used. When a device is used with the change form of the `scconf` command, the device's `autogen` property is reset, or set to false, unless `autogen=true` is also specified.

EXAMPLES

EXAMPLE 1 Using `scconf` Commands

The following `scconf` commands create a rawdisk device group, change the order of the potential primary nodes, change preference and failback policy, change the desired number of secondaries, and remove the rawdisk device group from the cluster configuration.

```
phys-host# scconf -a -D type=rawdisk,name=rawdisk_groupname,
nodelist=host1:host2:host3,preferenced=false,failback=enabled,
numsecondaries=,globaldev=d1,globaldev=d2

phys-host# scconf -a -D type=rawdisk,name=rawdisk_groupname,
nodelist=host1,globaldev=d1,globaldev=d2,localonly=true,
globaldev=d1,globaldev=d2

phys-host# scconf -c -D name=rawdisk_groupname,
nodelist=host3:host2:host1,preferenced=true,failback=disabled,
numsecondaries=2,globaldev=d4,globaldev=d5

phys-host# scconf -c -D name=rawdisk_groupname,localonly=true

phys-host# scconf -r -D name=rawdisk_groupname

phys-host# scconf -r -D name=rawdisk_groupname,nodelist=node1,node2
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcsu

SEE ALSO

`scconf`(1M), `attributes`(5)

NAME

scconf_dg_sds – change Solstice Disksuite disk device group configuration.

SYNOPSIS

scconf -c -D [*generic_options*]

DESCRIPTION

A Solstice DiskSuite disk device group is defined by a name, the nodes upon which this group can be accessed, a global list of devices in the diskset, and a set of properties used to control actions such as potential primary preference and failback behavior.

For Solstice DiskSuite disk device groups, only one diskset may be assigned to a disk device group, and the group name must always match the name of the diskset itself.

In Solstice DiskSuite, a multihosted or shared device is a grouping of two or more hosts and disk drives that are accessible by all hosts, and that have the same device names on all hosts. This identical device naming requirement is achieved by using the raw disk devices to form the diskset. The Disk ID pseudo driver (DID) allows multihosted devices to have consistent names across the cluster. Only hosts already configured as part of a diskset itself can be configured into the `nodelist` of a Solstice DiskSuite device group. At the time drives are added to a shared diskset, they must not belong to any other shared diskset.

The Solstice DiskSuite `metaset`(1M) command creates the diskset, which also initially creates and registers it as a Solstice DiskSuite device group. Next, you must use the `scconf(1M)` command to set the node preference list, the `preferenced` and `failback` suboptions, and change the desired number of secondaries.

If you want to change the order of node preference list or the failback mode, you must specify all the nodes that currenly exist in the device group in the `nodelist`. In addition, if you are changing the the order of node preference, you must also set the `preferenced` suboption to `true`.

If you do not specify the `preferenced` suboption with the “change”, the already established `true` or `false` setting is used.

You cannot use the `scconf` command to remove the Solstice DiskSuite device group from the cluster configuration. Use the Solstice DiskSuite `metaset` command instead. You remove a disk device group by removing the Solstice DiskSuite diskset.

OPTIONS

See `scconf(1M)` for the list of supported generic options. See `metaset`(1M) for the list of `metaset` related commands to create and remove disksets and disk device groups.

Only one action option is allowed in the command. The following action options are supported.

`-c` Change the ordering of the node preference list, change preference and failback policy, and change the desired number of secondaries.

EXAMPLES

EXAMPLE 1 Creating and Registering a Diskset

The following `metaset` commands create a diskset and register the diskset as a Solstice DiskSuite device group.

EXAMPLE 1 Creating and Registering a Diskset *(Continued)*

Next, the `scconf` command is used to specify the order of the potential primary nodes for the device group, change the preferenced and failback options, and change the desired number of secondaries.

```
phys-host# metaset -s diskset1 -a -h host1 host2

phys-host# scconf -c -D name=diskset1,nodelist=host2:host1,
preferenced=true,failback=disabled,numsecondaries=1
```

SEE ALSO scconf(1M), `metaset`(1M), `attributes`(5)

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWcsu`

NAME

scconf_dg_svm – change Solaris Volume Manager disk device group configuration.

SYNOPSIS

scconf -c -D [*generic_options*]

DESCRIPTION

A Solaris Volume Manager disk device group is defined by a name, the nodes upon which this group can be accessed, a global list of devices in the diskset, and a set of properties used to control actions such as potential primary preference and failback behavior.

For Solaris Volume Manager disk device groups, only one diskset may be assigned to a disk device group, and the group name must always match the name of the diskset itself.

In Solaris Volume Manager, a multihosted or shared device is a grouping of two or more hosts and disk drives that are accessible by all hosts, and that have the same device names on all hosts. This identical device naming requirement is achieved by using the raw disk devices to form the diskset. The Disk ID pseudo driver (DID) allows multihosted devices to have consistent names across the cluster. Only hosts already configured as part of a diskset itself can be configured into the `nodelist` of a Solaris Volume Manager device group. At the time drives are added to a shared diskset, they must not belong to any other shared diskset.

The Solaris Volume Manager `metaset`(1M) command creates the diskset, which also initially creates and registers it as a Solaris Volume Manager device group. Next, you must use the `scconf(1M)` command to set the node preference list, the `preferenced`, `failback` and `numsecondaries` suboptions.

If you want to change the order of node preference list or the failback mode, you must specify all the nodes that currenly exist in the device group in the `nodelist`. In addition, if you are changing the the order of node preference, you must also set the `preferenced` suboption to `true`.

If you do not specify the `preferenced` suboption with the “change”, the already established `true` or `false` setting is used.

You cannot use the `scconf` command to remove the Solaris Volume Manager device group from the cluster configuration. Use the Solaris Volume Manager `metaset` command instead. You remove a disk device group by removing the Solaris Volume Manager diskset.

OPTIONS

See `scconf(1M)` for the list of supported generic options. See `metaset`(1M) for the list of `metaset` related commands to create and remove disksets and disk device groups.

Only one action option is allowed in the command. The following action options are supported.

`-c` Change the ordering of the node preference list, change preference and failback policy, and change the desired number of secondaries.

EXAMPLES

EXAMPLE 1 Creating and Registering a Diskset

The following `metaset` commands create a diskset and register the diskset as a Solaris Volume Manager device group.

Next, the `scconf` command is used to specify the order of the potential primary nodes for the device group, change the preferenced and failback options, and change the desired number of secondaries.

```
phys-host# metaset -s diskset1 -a -h host1 host2

phys-host# scconf -c -D name=diskset1,nodelist=host2:host1,
preferenced=true,failback=disabled,numsecondaries=1
```

SEE ALSO

`scconf(1M)`, `metaset`(1M), `attributes`(5)

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWcsu`

NAME

scconf_dg_vxvm – add, change, or update VxVM device group configuration.

SYNOPSIS

scconf -a -D type=vxvm,*devicegroup-options*

scconf -c -D *devicegroup-options* [,sync]

scconf -r -D name=*devicegroupname*

DESCRIPTION

The scconf_dg_vxvm(1M) command is used to add, change, and remove the VERITAS Volume Manager (VxVM) disk device groups to the Sun Cluster device-groups configuration.

The add (-a) option adds a new VxVM disk device group to the Sun Cluster device-groups configuration. With this option you define a name for the new device group, specify the nodes on which this group can be accessed, and specify a set of properties used to control actions.

For VxVM disk device groups, you can only assign one VxVM disk group to a disk device group, and the disk device group name must always match the name of the VxVM disk group. You cannot create a VxVM disk device group unless you first import the corresponding VxVM disk group on one of the nodes in that device's nodelist.

Before you can add a node to a VxVM disk device group, every physical disk in the disk group must be physically ported to that node. After you register the disk group as a VxVM disk device group, you must first deport the disk group from the current node owner and turn off the auto-import flag for the disk group.

To create a VxVM disk device group for a disk group, you must run the scconf(1M) command from the same node where the disk group was created.

The scconf change (-c) command changes the order of the potential primary node preference, to enable or disable failback, to add more global devices to the device group, and to change the desired number of secondaries.

To change the order-of-node preference list from false to true, you must specify in thenodelist all the nodes that currently exist in the device group. You must also set the preferenced suboption to true.

If you do not specify the preferenced suboption with the change form of the command, the already established true or false setting is used.

The sync option is used to synchronize the clustering software with VxVM disk-group volume information. The sync option is only valid with the change form of the command. Use the sync option whenever you add or remove a volume from a device group.

The remove (-r) option removes a VxVM device group from the Sun Cluster device-groups configuration. You can also use this form of command to remove the nodes from the VxVM disk device group configuration.

OPTIONS

See the scconf(1M) man page for the list of supported generic device-group options.

The following action options describe the actions that the command performs. Only one action option is allowed in the command.

The following action options are supported:

`-a` Add a VxVM device group to the cluster configuration.

`-c` Change the ordering of the node preference list, change preference and failback policy, and change the desired number of secondaries.

`-r` Remove the specified VxVM device group from the cluster.

EXAMPLES

EXAMPLE 1 Using `scconf` Commands

The following `scconf` commands create a VxVM device group, change the order of the potential primary nodes, change the preference and failback policy for the device group, change the desired number of secondaries, and remove the VxVM device group from the cluster configuration.

```
host1# scconf -a -D type=vxvm,name=diskgrp1,
nodelist=host1:host2:host3,preferenced=false,failback=enabled
host1# scconf -c -D name=diskgrp1,
nodelist=host2:host1:host3,preferenced=true,failback=disabled,
numsecondaries=2
host1# scconf -r -D name=diskgrp1,nodelist=node1
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

`scconf`(1M), `attributes`(5)

NAME

scconf_quorum_dev_netapp_nas – add and remove shared Network Appliance network-attached storage (NAS) quorum devices and change various NAS cluster quorum configuration properties or states.

SYNOPSIS

scconf {-a| -c | -r } -q name=*devicename otheroptions*

DESCRIPTION

A Network Appliance NAS device can be configured as a quorum device for Sun Cluster. The NAS configuration information consists of:

- a device name, which must be unique across quorum devices
- a filer name, which defaults to the device name if not specified
- a LUN ID, which defaults to 0 if not specified

To provide support for NAS devices as quorum devices, the administrator must install the quorum device support module provided by Network Appliance. If this module is not available, scconfprevents the addition of the quorum device. See *Sun Cluster With Network-Attached Storage Devices Manual for Solaris OS* for instructions about obtaining the support module.

Additionally, the iSCSI license must be valid for the Network Appliance device.

OPTIONS

The following options can be used for NAS quorum devices. See scconf(1M) for the list of supported generic options. See scconf_quorum_dev_netapp_nas(1M) for options that are specific to shared disk quorum devices.

The add and remove forms of the command are used to add and remove NAS quorum devices to or from the configuration. The change form of the command is used for changing various properties of cluster quorum configuration.

Before you add a quorum device, you must set up and configure the device and the logical unit number (LUN) on the device to be configured as a quorum device. For detailed procedures, see your Network Appliance documentation. For Sun Cluster requirements of device setup, see *Sun Cluster With Network-Attached Storage Devices Manual for Solaris OS*. After the quorum device is added, you cannot change the type.

Add a NAS quorum device:

-q -a name=*devicename*,type=netapp_nas[,filer=*filer-name*][,lun_id=*0*]

Change a NAS quorum device's configuration:

-q -c name=*devicename*,{maintstate | reset}

Remove a NAS quorum device:

-q -r name=*devicename*

The -q option supports the following Network Appliance NAS-specific suboptions:

filer=*filer-name*
: Specifies the name of the device on the network that you can use to access the NAS device when you are using rsh or telnet.

lun_id=*0*
: Specifies the LUN ID on the NAS device that will be a NAS quorum device. The LUN ID defaults to 0.

When `scconf` is interrupted or fails while performing quorum-related operations, quorum configuration information can become inconsistent in the cluster configuration database. If an inconsistency occurs, either run the same `scconf` command again or run it with the `reset` option to reset the quorum information.

EXAMPLES

EXAMPLE 1 Adding Network Appliance NAS Quorum Devices

The following `scconf` command adds the Network Appliance NAS quorum device `qd1`.

```
-a -q name=qd1,type=netapp_nas,filer=nas1.sun.com,lun_id=0
```

EXAMPLE 2 Removing Network Appliance NAS Quorum Devices

The following `scconf` command removes the Network Appliance NAS quorum device `qd1`.

```
-r -q name=qd1
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

`scconf`(1M), `scconf-quorum-dev-scsi`(1M)

NAME

scconf_quorum_dev_scsi – Add and remove shared SCSI quorum devices and change various SCSI cluster quorum configuration properties or states.

SYNOPSIS

scconf {-a| -c | -r } -q globaldev=*devicename otheroptions*

scconf {-a| -c | -r } -q name=*devicename otheroptions*

DESCRIPTION

A SCSI quorum device is considered to be any Sun Cluster supported, attached storage that is connected to two or more nodes of the cluster. The device must be managed by DID, and the device name that is provided must be a DID device name.

The SCSI quorum device has no other properties that can be specified.

OPTIONS

The following options are specific to shared disk quorum devices. See scconf(1M) for the list of supported generic options. See scconf_quorum_dev_netapp_nas(1M) for options that are specific to NAS quorum devices.

The add and remove forms of the command are used to add and remove shared quorum devices to or from the configuration. The change form of the command is used for changing various properties of cluster quorum configuration. The -q *quorum-options* available for each of the three forms of the command can be used to change the cluster quorum configuration are as follows:

Add a shared quorum device:

-q -a globaldev=*devicename*[, node=*node*,node=*node*[, ...]]
or

-q -a name= *devicename*,type=scsi
or

-q -a autoconfig[,noop]

Change a property or state of quorum configuration:

-q -c globaldev=*devicename*,{maintstate | reset}
or

-q -c autoconfig[,noop]

Remove a shared quorum device:

-q -r globaldev=*devicename*
or

-q -r name=*devicename*

autoconfig
When used with the add form of the command, automatically chooses and assigns one quorum device in the two-node cluster. The quorum device is chosen from the available devices. If a quorum device is already configured, the command aborts.

When used with the change form of the command, automatically chooses and assigns one device that replaces all existing quorum devices in the two-node cluster. The quorum device is chosen from the available devices.

All available devices in the cluster must be qualified to be a quorum device. The `autoconfig` suboption does not assess whether an available device is qualified to be a quorum device.

If the cluster contains more than two nodes, the `autoconfig` suboption makes no changes to the quorum configuration. Do not use the `autoconfig` suboption if you intend to configure a NAS device as quorum.

[,`noop`]
Is valid with the `autoconfig` suboption. The command prints to standard output the list of quorum devices that the `autoconfig` suboption would add or change. The `autoconfig,noop` suboption makes no changes to the quorum configuration.

When `scconf` is interrupted or fails while performing quorum-related operations, quorum configuration information can become inconsistent in the cluster configuration database. If an inconsistency occurs, either run the same `scconf` command again or run it with the `reset` option to reset the quorum information.

With the `add` form of the command, if a `name` is specified without a `node` list, the quorum device is added with a port defined for every node to which the device is attached. But, if a `node` list is given, at least two nodes must be provided, and each node in the list must be ported to the device.

EXAMPLES

EXAMPLE 1 Adding SCSI Quorum Devices

The following `scconf` commands adds a SCSI quorum device.

```
-a -q globaldev=/dev/did/rdsk/d4s2
    or
-a -q name=/dev/did/rdsk/d4s2,type=scsi
```

EXAMPLE 2 Changing SCSI Quorum Devices

The following `scconf` command changes a SCSI quorum device configuration.

```
-c -q globaldev=/dev/did/rdsk/d4s2,reset
    or
-c -q name=/dev/did/rdsk/d4s2,reset
```

EXAMPLE 3 Removing SCSI Quorum Devices

The following `scconf` command removes the SCSI quorum device. `qd1`.

```
-r -q globaldev=qd1
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO scconf(1M), scconf_quorum_dev_netapp_nas(1M)

NAME

scconf_transp_adap_bge – configure the bge transport adapter

DESCRIPTION

`bge` adapters may be configured as cluster transport adapters. These adapters may only be used with the `dlpi` transport type. The `bge` adapter is VLAN capable.

The `bge` adapter connects to a transport junction or to another `bge` adapter on a different node. In either case, the connection is made through a transport cable.

When a transport junction is used and the endpoints of the transport cable are configured using the `scconf` command, the `scinstall` command, or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the junction.

Refer to `scconf`(1M) for more configuration details.

The default is to set the port name to the node ID hosting the adapter at the other end of the cable.

There are no user configurable properties for cluster transport adapters of this type.

SEE ALSO

`scinstall`(1M), `scconf`(1M)

NAME

scconf_transp_adap_ce – configure the ce Sun Ethernet transport adapter

DESCRIPTION

`ce` adapters can be configured as cluster transport adapters. These adapters can be used with transport types `dlpi`. The `ce` adapter is VLAN capable.

A `ce` adapter connects to a transport junction or to another `ce` adapter on a different node. In either case, the connection is made through a transport cable.

When a transport junction is used and the endpoints of the transport cable are configured using `scconf`(1M), `scinstall`(1M) or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the junction.

The default is to set the port name to the node ID hosting the adapter at the other end of the cable.

There are no user configurable properties for cluster transport adapters of this type.

SEE ALSO

`scconf`(1M), `scinstall`(1M)

NAME

scconf_transp_adap_e1000g – configure the Intel PRO/1000 network adapter

DESCRIPTION

`e1000g` Intel PRO/1000 network adapters can be configured. These adapters can only be used with transport type `dlpi`.

The `e1000g` based network adapter connects to a transport junction or to another Ethernet adapter on a different node. In either case, the connection is made through a transport cable.

When a transport junction is used and the endpoints of the transport cable are configured using `scconf(1M)`, `scinstall(1M)`, or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the junction.

The default is to set the port name to the node identifier that hosts the adapter at the other end of the cable.

Refer to `scconf(1M)` for more configuration details.

There are no user configurable properties for cluster transport adapters of this type.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

SEE ALSO

`scconf(1M)`, `scinstall(1M)`, `e1000g`(7D)

NAME

scconf_transp_adap_eri – configure the eri transport adapter

DESCRIPTION

`eri` Ethernet adapters can be configured as cluster transport adapters. These adapters can only be used with transport type `dlpi`.

The `eri` Ethernet adapter connects to a transport junction or to another Ethernet adapter on a different node. In either case, the connection is made through a transport cable.

When a transport junction is used and the endpoints of the transport cable are configured using `scconf(1M)`, `scinstall(1M)`, or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the junction.

The default is to set the port name to the node ID hosting the adapter at the other end of the cable.

Refer to `scconf(1M)` for more configuration details.

There are no user configurable properties for cluster transport adapters of this type.

SEE ALSO

`scconf(1M)`, `scinstall(1M)`, `eri`(7D)

NAME

scconf_transp_adap_ge – configure the Gigabit Ethernet (ge) transport adapter

DESCRIPTION

`ge` adapters can be configured as cluster transport adapters. These adapters can only be used with transport type `dlpi`.

The `ge` adapter connects to a transport junction or to another `ge` adapter on a different node. In either case, the connection is made through a transport cable.

When a transport junction is used and the endpoints of the transport cable are configured using `scconf(1M)`, `scinstall(1M)`, or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the junction.

The default is to set the port name to the node `ID` hosting the adapter at the other end of the cable.

Refer to `scconf(1M)` for more configuration details.

There are no user configurable properties for cluster transport adapters of this type.

SEE ALSO

`scconf(1M)`, `scinstall(1M)`

NAME

scconf_transp_adap_hme – configure the hme transport adapter

DESCRIPTION

`hme` Ethernet adapters can be configured as cluster transport adapters. These adapters may only be used with transport type `dlpi`.

The `hme` Ethernet adapter connects to a transport junction or to another Ethernet adapter on a different node. In either case, the connection is made through a transport cable.

When a transport junction is used and the endpoints of the transport cable are configured using `scconf(1M)`, `scinstall(1M)`, or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the junction.

The default is to set the port name to the node ID hosting the adapter at the other end of the cable.

Refer to `scconf(1M)` for more configuration details.

There are no user configurable properties for cluster transport adapters of this type.

SEE ALSO

`scconf(1M)`, `scinstall(1M)`, `hme`(7D)

NAME

scconf_transp_adap_ibd – configure the InfiniBand (ibd) transport adapter

DESCRIPTION

`ibd` adapters can be configured as cluster transport adapters. These adapters can only be used with transport type `dlpi`.

The `ibd` adapter connects to an InfiniBand transport junction. The connection is made through a transport cable.

When the endpoints of the transport cable are configured by using `scconf`(1M), `scinstall`(1M), or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the junction.

The default is to set the port name to the node `ID` that hosts the adapter at the other end of the cable.

Refer to `scconf`(1M) for more configuration details.

There are no user-configurable properties for cluster transport adapters of this type.

SEE ALSO

`scconf`(1M), `scconf_transp_jct_ibswitch`(1M), `scinstall`(1M)

NAME

scconf_transp_adap_qfe – configure the qfe transport adapter

DESCRIPTION

`qfe` Ethernet adapters can be configured as cluster transport adapters. These adapters can only be used with transport type `dlpi`.

The `qfe` Ethernet adapter connects to a transport junction or to another Ethernet adapter on a different node. In either case, the connection is made through a transport cable.

When a transport junction is used and the endpoints of the transport cable are configured using `scconf(1M)`, `scinstall(1M)`, or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the junction.

The default is to set the port name to the node ID hosting the adapter at the other end of the cable.

Refer to `scconf(1M)` for more configuration details.

There are no user configurable properties for cluster transport adapters of this type.

SEE ALSO

`scconf(1M)`, `scinstall(1M)`, `qfe`(7D)

NAME

scconf_transp_adap_sci – configure the sci cluster transport adapter

DESCRIPTION

SCI-PCI adapters can be configured as cluster transport adapters. These adapters can be used with the `dlpi` transport type.

The adapter name is `sci`*N*, for example, `sci0`. Do not use `scid`*N* as the adapter name.

An `sci` adapter can only be connected to another `sci` adapter or to an SCI switch. When an `sci` adapter is connected to an SCI switch, it is important that you specify the correct port name when referring to a port on the switch as an endpoint argument to the `scconf`(1M) or `scinstall`(1M) utility. The port name must match the port number on the SCI switch (the number printed on the switch itself). Failure to give the correct port name could result in the `scconf` or `scinstall` utility failing. The result of providing an incorrect port name will be the same as you would see if the cable between the adapter and the switch were removed.

There are no user-configurable properties for cluster transport adapters of this type.

SEE ALSO

`scconf`(1M), `scconf_transp_jct_dolphinswitch`(1M), `scinstall`(1M)

NAME

scconf_transp_adap_wrsm.1m – configure the wrsm transport adapter

DESCRIPTION

`wrsm` adapters may be configured as cluster transport adapters. These adapters can only be used with transport types `dlpi`.

The `wrsm` adapter connects to a transport junction or to another `wrsm` adapter on a different node. In either case, the connection is made through a transport cable.

Although you can connect the `wrsm` adapters directly by using a point-to-point configuration, Sun Cluster software requires that you specify a *virtual* transport junction. For example, if `node1:wrsm1` is connected to `node2:wsrm1` directly through a cable, you must specify the following configuration information.

```
node1:wrsm1 <--cable1--> Transport Junction sw_wrsm1 <--cable2--> node2:wrsm1
```

The transport junction, whether a virtual switch or a hardware switch, must have a specific name. The name must be `sw_wrsm`*N* where the adapter is `wrsm`*N*. This requirement reflects a Wildcat restriction that requires that all `wrsm` controllers on the same Wildcat network have the same instance number.

When a transport junction is used and the endpoints of the transport cable are configured using `scconf(1M)` , `scinstall(1M)`, or other tools, you are asked to specify a port name on the transport junction. You can provide any port name, or accept the default, as long as the name is unique for the transport junction.

The default sets the port name to the node ID that hosts the adapter at the other end of the cable.

Refer to `scconf(1M)` for more configuration details.

There are no user configurable properties for cluster transport adapters of this type.

SEE ALSO

`scconf(1M)`, `scinstall(1M)`, `wrsmconf`(1M), `wrsmstat`(1M), `wrsm`(7D), `wrsmd`(7D)

NAME

scconf_transp_jct_dolphinswitch – configure the Dolphin cluster transport junction

DESCRIPTION

`SCI` switches may be used as cluster transport junctions. They are of junction type `switch`.

The Dolphin `SCI` switch is used with `SCI` adapters. The ports of a Dolphin `SCI` switch are numbered (printed on the switch itself). The port number should be used as the name of the port. It is important that you specify the correct port name when referring to a port on the switch as an endpoint argument to `scconf`(1M) or `scinstall`(1M). Failure to give the correct port name (which must be the same as the port number that appears on the switch), could result in `scconf` or `scinstall` failing or an operation running on a wrong port. This might bring down the cluster or prevent a node from coming up in clustered mode.

There are no user configurable properties on the Dolphin `SCI` switch.

SEE ALSO

`scconf`(1M), `scinstall`(1M)

NAME

scconf_transp_jct_etherswitch – configure an Ethernet cluster transport junction

DESCRIPTION

Ethernet switches can be configured as cluster transport junctions. They are of junction type `switch`. There are no user configurable properties.

SEE ALSO

`scconf`(1M)

NAME

scconf_transp_jct_ibswitch – configure an InfiniBand cluster transport junction

DESCRIPTION

InfiniBand switches can be configured as cluster transport junctions. They are of junction type `switch`. There are no user configurable properties.

SEE ALSO

`scconf`(1M), `scconf_transp_adap_ibd`(1M)

NAME

scdidadm – device identifier configuration and administration utility wrapper

SYNOPSIS

/usr/cluster/bin/scdidadm -c

/usr/cluster/bin/scdidadm -C

/usr/cluster/bin/scdidadm -r

/usr/cluster/bin/scdidadm -R {*path* | *instance_number* | all}

/usr/cluster/bin/scdidadm -l | -L [-h] [-o *fmt*] ...
[*path* | *instance_number*]

/usr/cluster/bin/scdidadm [-u] [-i]

/usr/cluster/bin/scdidadm -U

/usr/cluster/bin/scdidadm -v

DESCRIPTION

The scdidadm utility administers the device identifier (DID) pseudo device driver did(7).

The scdidadm utility performs the following primary operations:

- Creates driver configuration files
- Modifies entries in the file
- Loads the current configuration into the kernel
- Lists the mapping between device entries and did driver instance numbers

The startup script /etc/init.d/bootcluster uses the scdidadm utility to initialize the did driver. You can also use scdidadm to update or query the current device mapping between the devices present and the corresponding device identifiers and did driver instance numbers.

The devfsadm(1M) command creates the file system device entry points.

OPTIONS

The following options are supported:

-c
: Performs a consistency check against the kernel representation of the devices and the physical devices. On failing a consistency check, an error message is displayed. The process continues until all devices have been checked.

  You need solaris.cluster.device.read RBAC authorization to use this command option. See rbac(5).

-C
: Removes all did references to underlying devices that have been detached from the current node. Specify this option after the Solaris device commands have been used to remove references to nonexistent devices on the cluster nodes.

  You can only use this option from a node that is booted in cluster mode.

You need `solaris.cluster.device.modify` RBAC authorization to use this command option. See `rbac`(5).

`-h` Prints a header when listing device mappings. This option is meaningful only when used with the `-l` and `-L` options.

`-i` Initializes the `did` driver. Use this option if you want to enable I/O requests to the `did` driver.

You need `solaris.cluster.device.modify` RBAC authorization to use this command option. See `rbac`(5).

`-l` Lists the local devices in the DID configuration file. The output of this command can be customized using the `-o` option. When no `-o` options are specified, the default listing displays the *instance* number, the local *fullpath*, and the *fullname*.

You need `solaris.cluster.device.read` RBAC authorization to use this command option. See `rbac`(5).

`-L` Lists all the paths, including those on remote hosts, of the devices in the DID configuration file. The output of this command can be customized using the `-o` option. When no `-o` options are specified, the default listing displays the *instance* number, all local and remote *fullpath* strings, and the *fullname*.

You need `solaris.cluster.device.read` RBAC authorization to use this command option. See `rbac`(5).

`-o` *fmt* Lists the devices currently known to the `did` driver according to the format specification *fmt*. Multiple `-o` options can be specified. The *fmt* specification is interpreted as a comma-separated list of format option arguments. This option is meaningful only when used with the `-l` and `-L` options. The available format option arguments are the following:

instance Prints the instance number of the device known by the `did` driver, for example, `1`.

path Prints the physical path name of the device associated with this device identifier, for example, `/dev/rdsk/c0t3d0`.

fullpath Prints the full physical path name of the device that is associated with this device identifier. This path name includes the host, for example, `phys-hostA:/dev/rdsk/c0t3d0`.

host With the `-L` option, prints the names of all hosts that have connectivity to the specified device, one per line. With the `-l` option, prints the name of the local host that has connectivity to the specified device.

name Prints the DID name of the device associated with this device identifier, for example, `d1`.

fullname Prints the full DID path name of the device associated with this device identifier, for example, `/dev/did/rdsk/d1`.

diskid Prints the hexadecimal representation of the device identifier associated with the instance of the device being listed.

asciidiskid Prints the ASCII representation of the device identifier associated with the instance of the device being listed.

`-r` Reconfigures the database. When you specify this option, a thorough search of the `rdsk` and `rmt` device trees is conducted. A new instance number is assigned for all device identifiers that were not recognized before. A new path is added for each newly recognized device.

You can only use this option from a node that is booted in cluster mode.

You need `solaris.cluster.device.modify` RBAC authorization to use this command option. See `rbac`(5).

`-R` {*path* | *instance_number* | all} Performs a repair procedure on a particular device instance. The argument to this command can be either a particular physical device *path* that has been replaced with a new device, or the *instance_number* of the device that was just replaced. When used with the `all` keyword, the `scdidadm` utility updates the configuration data of all devices connected to the node.

You can only use this option from a node that is booted in cluster mode.

You need `solaris.cluster.device.modify` RBAC authorization to use this command option. See `rbac`(5).

`-u` Loads the device identifier configuration table into the kernel. This option loads all the currently known configuration information about device paths and their corresponding instance numbers into the kernel.

You need `solaris.cluster.device.modify` RBAC authorization to use this command option. See `rbac`(5).

`-U` Converts an existing `/etc/did.conf` file into a set of Cluster Configuration Repository (CCR) tables. If the tables already exist, this command fails.

You need `solaris.cluster.device.modify` RBAC authorization to use this command option. See `rbac`(5).

`-v` Prints the version number of this program.

EXAMPLES

EXAMPLE 1 Adding Devices Attached to the Local Host to the CCR

```
% scdidadm -r
```

EXAMPLE 2 Listing the Physical Path of the Device

The following example lists the physical path of the device that corresponds to instance `2` of the `did` driver:

```
% scdidadm -l -o path 2
/dev/dsk/c1t4d0
```

EXAMPLE 3 Specifying Multiple Format Options

You can specify multiple format option arguments in either of the following ways:

```
% scdidadm –l –o path –o name 2

% scdidadm -l -o path,name 2
```

In either example, the output might look like this:

```
/dev/dsk/c1t4d0 d1
```

EXAMPLE 4 Performing a Repair Procedure

The following example performs the repair procedure for a particular device path. The device `/dev/dsk/c1t4d0` has been replaced with a new device with which a new device identifier is associated. The database is updated to show that this new device identifier corresponds to the instance number that was previously associated with the old device identifier:

```
% scdidadm -R c1t4d0
```

EXAMPLE 5 Performing a Repair Procedure

An alternative method of performing a repair procedure is to use the instance number associated with the device path. For example, if the instance number for the device `c1t4d0` in the previous example is `2`, then the following syntax performs the same operation as the previous example:

```
% scdidadm -R 2
```

EXIT STATUS

The following exit values are returned:

`0` The command completed successfully.

`1` An error occurred.

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu
Interface Stability	Evolving

SEE ALSO `devfsadm`(1M), `attributes`(5), `did`(7)

Sun Cluster 3.1 System Administration Guide

NOTES Each multiported tape drive or CD-ROM drive appears in the namespace once per physical connection.

NAME

scdpm – Disk-path monitoring administration command

SYNOPSIS

```
scdpm -m [node|all]:<[/dev/did/rdsk/]d- | [/dev/rdsk/]c-t-d- | all>
scdpm -u [node|all]:<[/dev/did/rdsk/]d- | [/dev/rdsk/]c-t-d- | all>
scdpm -p [-F] [node|all]:<[/dev/did/rdsk/]d- | [/dev/rdsk/]c-t-d- | all>
scdpm -f filename
```

DESCRIPTION

The `scdpm` command manages the disk-path monitoring daemon in a cluster environment. This command is used to monitor and unmonitor disk paths. You can also use the `scdpm` command to display the status of disk paths. All of the accessible disk paths in the cluster or on a specific node are printed to the standard output. The `scdpm` command must be run from a cluster node that is online in cluster mode.

You can specify either a global name or a UNIX name when you monitor a new disk path. Additionally, you can force the daemon to reread the entire disk configuration.

OPTIONS

The following options are supported.

`-m`
Monitor the new disk path that is specified by *node:disk path*. If the node name is not specified, `all` is the default option.

`-u`
Unmonitor a disk path. The daemon on each node stops monitoring the specified path.

`-p`
Print the current status of a specified disk path from all the nodes that are attached to the storage. With the `-F` option, `scdpm` prints the faulty disk paths in the cluster. If the node name is not specified, `all` is the default option. The status can be `Ok` , `Fail`, `Unmonitored`, or `Unknown`.

Note – You need `solaris.cluster.device.read` RBAC authorization to use this command with the `-p` option. See `rbac`(5).

`-f` *file name*
Read the list of disk paths to monitor or unmonitor for a specified file name. The file must list the command to monitor or unmonitor, node-name and disk-path name. The commands are `m` for monitor, and `u` for unmonitor. The command must be followed by a space. The node-name and disk-path name should be separated by a colon.

```
syntax in command file:
[u,m] [node|all]:<[/dev/did/rdsk/]d- | [/dev/rdsk/]c-t-d- | all>

command file entry
u schost-1:/dev/did/rdsk/d5
m schost-2:all
```

Note – You need `solaris.cluster.device.admin` RBAC authorization to use this command with the `-m`, `-u` and `-f` options. See See `rbac`(5).

EXIT STATUS

The following exit values are returned:

`>0`	An error occurred. Error messages are displayed on the standard error.
`0`	Successful completion.
`1`	Complete failure.
`2`	Partial failure.

Note – The disk path is represented by a node name and a disk name. The node name must be the hostname or the word `all` to address all of the nodes in the cluster. The disk name must be the global disk name, a UNIX path name, or the word `all` to address all the disks in the node. The disk name can be either the full global path name or just the disk name, for example `/dev/did/dsk/d3` or `d3`. The disk name can also be the full UNIX path name, for example `/dev/rdsk/c0t0d0s0`.

Disk-path status changes are logged by using the `syslogd` `LOG_INFO` facility level. All failures are logged by using `LOG_ERR` facility level.

EXAMPLES

EXAMPLE 1 Monitoring All Disk Paths in the Cluster Infrastructure

The following command forces the daemon to monitor all disk paths in the cluster infrastructure.

```
# scdpm -m all
```

EXAMPLE 2 Monitoring a New Disk Path

The following command monitors a new disk path. In the following example, all nodes monitor `/dev/did/dsk/d3` where this path is valid.

```
# scdpm -m /dev/did/dsk/d3
```

EXAMPLE 3 Monitoring a Disk Path on a Single Node

The following command monitors a new path on a single node. The daemon on the `schost-2` node monitors paths to the `/dev/did/dsk/d4` and `/dev/did/dsk/d5` disks.

```
# scdpm -m schost-2:d4 -m schost-2:d5
```

EXAMPLE 4 Printing All Disk Paths and the Status

The following command prints all disk paths in the cluster and their status.

```
# scdpm -p all:all
         schost-1:/dev/did/dsk/d4   Ok
         schost-1:/dev/did/dsk/d3   Ok
         schost-2:/dev/did/dsk/d4   Fail
         schost-2:/dev/did/dsk/d3   Ok
         schost-2:/dev/did/dsk/d5   Unmonitored
         schost-2:/dev/did/dsk/d6   Ok
```

EXAMPLE 5 Printing All of the Failed Disk Paths

The following command prints all of the failed disk paths on the `schost-2` node.

```
# scdpm -p -F all
         schost-2:/dev/did/dsk/d4   Fail
```

EXAMPLE 6 Printing the Status of all Disk Paths From a Single Node

The following command prints the disk path and the status for disks that are monitored on the `schost-2` node.

```
# scdpm -p schost-2:all
         schost-2:/dev/did/dsk/d4   Fail
         schost-2:/dev/did/dsk/d3   Ok
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes.

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu

SEE ALSO

`scconf`(1M), `scdidadm`(1M)

Sun Cluster 3.1 System Administration Guide

NAME

scgdevs – global devices namespace administration script

SYNOPSIS

/usr/cluster/bin/scgdevs

DESCRIPTION

The `scgdevs` utility manages the global devices namespace. The global devices namespace is mounted under `/global` and consists of a set of logical links to physical devices. As `/dev/global` is visible to each node of the cluster, each physical device is visible across the cluster. This fact means that any disk, tape, or CD-ROM that is added to the global devices namespace can be accessed from any node in the cluster.

The `scgdevs` command allows the administrator to attach new global devices (for example, tape drives, CD-ROM drives, and disk drives) to the global devices namespace without requiring a system reboot. The `drvconfig`(1M) and `devlinks`(1M) commands must be executed prior to running the `scgdevs` script.

Alternatively, a reconfiguration reboot can be used to rebuild the global namespace and attach new global devices. See `boot`(1M).

This script must be run from a node that is a current cluster member. If this script is run from a node that is not a cluster member, the script exits with an error code and leaves the system state unchanged.

You need `solaris.cluster.system.modify` RBAC authorization to use this command. See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

EXIT STATUS

The following exit values are returned:

`0`	The command completed successfully.
nonzero	An error occurred. Error messages are displayed on the standard output.

FILES

`/devices`	Device nodes directory
`/global/.devices`	Global devices nodes directory
`/dev/md/shared`	SDS/Solaris Volume Manager metaset directory

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`boot`(1M), `devfsadm`(1M), `devlinks`(1M), `drvconfig`(1M), `scdidadm`(1M), `attributes`(5), `did`(7)

Sun Cluster System Administration Guide

NOTES

The `scgdevs` command, called from the local node, will perform its work on remote nodes asynchronously. Therefore, command completion on the local node does not necessarily mean it has completed its work clusterwide.

This document does not constitute an API. `/global/.devices` and `/devices` might not exist or might have different contents or interpretations in a future release. The existence of this notice does not imply that any other documentation that lacks this notice constitutes an API. This interface should be considered an unstable interface.

NAME

scinstall – install Sun Cluster software and initialize new cluster nodes

SYNOPSIS

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`
`-i [-k] [-d` *cdimage-dir*`] [ -s` *srvc*`[,...]] [-M` *patch-options*`]`
`[-F [-C` *clustername*`] [-T` *authentication-options*`] [-G [`*special* | *mount-point*`]`
`[-o` *only one*`]] [-A` *adapter-options*`] [-B` *junction-options*`] [-m` *cable-options*`]`
`[-w` *netaddr-options*`]]`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`
`-i [-k] [-d` *cdimage-dir*`] [ -s` *srvc*`[,...]] [-M` *patch-options*`]`
`[-N` *cluster-member* `[-C` *clustername*`] [-G {`*special* | *mount-point*`}]`
`[-A` *adapter-options*`] [-B` *junction-options*`] [-m` *cable-options*`]]`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`
`-a` *install-dir* `[-d` *cdimage-dir*`]`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`
`-c` *jumpstart-dir* `-h` *nodename* `[-d` *cdimage-dir*`] [ -s` *srvc*`[,...]]`
`[-M` *patch-options*`] [-F [-C` *clustername*`] [-G {`*special* | *mount-point*`}]]`
`[-T` *authentication-options* `[-A` *adapter-options*`] [-B` *junction-options*`]`
`[-m` *cable-options*`] [-w` *netaddr-options*`]]`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`
`-c` *jumpstart-dir* `-h` *nodename* `[-d` *cdimage-dir*`] [ -s` *srvc*`[,...]]`
`[-M` *patch-options*`] [-N` *cluster-member* `[-C` *clustername*`]`
`[-G {`*special* | *mount-point*`}] [-A` *adapter-options*`] [-B` *junction-options*`]`
`[-m` *cable-options*`]]`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`
`-u update [`*upgrade-options*`] [-M` *patch-options*`]`

`/usr/cluster/bin/scinstall -r [-N` *cluster-member*`] [-G` *mount-point*`]`

`scinstall -p [-v]`

DESCRIPTION

The `scinstall` command performs a number of Sun Cluster node initialization, installation, and upgrade tasks, as follows.

- The "install" form (`-i`) of `scinstall` installs and initializes a node as a new Sun Cluster member. It either establishes the first node in a new cluster (`-F`) or adds a node to an already-existing cluster (`-N`). Always run this form of the `scinstall` command from the node that is being installed or added to the cluster.
- The "set up install server" form (`-a`) of `scinstall` creates an *install-dir* on any Solaris machine from which the command is run and then copies a Sun Cluster installation media to that directory. Typically, you would create the target directory on an NFS server which has also been set up as a Solaris install server (see the `setup_install_server`(1M) man page).
- The "add install client" form (`-c`) of `scinstall` establishes the given *nodename* as a custom JumpStart client in the *jumpstart-dir* on the machine from which the command is run. Typically, the *jumpstart-dir* is located on an already-established

Solaris install server configured to JumpStart the Solaris *nodename* install client (see the `add_install_client`(1M) man page).

- The "remove" form (`-r`) of `scinstall` removes cluster configuration information and uninstalls Sun Cluster software from a cluster node.
- The "upgrade" form (`-u`) of `scinstall` upgrades a Sun Cluster node. Always run this form of the `scinstall` command from the node being upgraded.
- The "print release" form (`-p`) of `scinstall` prints release and package versioning information for the Sun Cluster software installed on the node from which the command is run.

Without options, the `scinstall` command attempts to run in interactive mode.

Run all forms of the `scinstall` command other than the "print release" form (`-p`) as superuser.

The `scinstall` command is located in the `Tools` directory on the Sun Cluster installation media. If the Sun Cluster installation media has been copied to a local disk, *media-mnt-pt* is the path to the copied Sun Cluster media image. The `SUNWscu` software package also includes a copy of the `scinstall` command.

OPTIONS

Basic Options

The following options direct the basic form and function of the command.

None of the following options can be combined on the same command line.

`-a`

Specifies the "set up install server" form of the `scinstall` command. This option is used to create an *install-dir* on any Solaris machine from which the command is run and then make a copy of the Sun Cluster media in that directory.

If the *install-dir* already exists, the `scinstall` command returns an error message. Typically, the target directory is created on an NFS server which has also been set up as a Solaris install server (see the `setup_install_server`(1M) man page).

`-c`

Specifies the "add install client" form of the `scinstall` command. This option establishes the given *nodename* as a custom JumpStart client in the *jumpstart-dir* on the machine from which you issued the command.

Typically, the *jumpstart-dir* is located on an already-established Solaris install server that is configured to JumpStart the *nodename* install client (see the `add_install_client`(1M) man page).

This form of the command enables fully-automated cluster installation from a JumpStart server by helping to establish each cluster node, or *nodename*, as a custom JumpStart client on an already-established Solaris JumpStart server. The command makes all necessary updates to the `rules` file in the given *jumpstart-dir*. In addition, special JumpStart `class` files and `finish` scripts that support cluster initialization are added to the *jumpstart-dir*, if they are not already installed. Configuration data that is used by the Sun Cluster-supplied `finish` script is established for each node that you set up by using this method.

Users can customize the Solaris `class` file that the `-c` option to the `scinstall` command installs by editing the file directly in the normal way. However, it is always important to ensure that the Solaris `class` file defines an acceptable Solaris installation for a Sun Cluster node. Otherwise, the installation might need to be restarted.

Both the `class` file and `finish` script installed by this form of the command are located in the following directory:

jumpstart-dir`/autoscinstall.d/3.1`

The `class` file is installed as `autoscinstall.class`, and the `finish` script is installed as `autoscinstall.finish`.

For each cluster *nodename* that you set up with the `-c` option as an automated Sun Cluster JumpStart install client, this form of the command sets up a configuration directory as the following:

jumpstart-dir`/autoscinstall.d/nodes/`*nodename*

Options for specifying Sun Cluster node installation and initialization are saved in files located in these directories. Never edit these files directly.

You can customize the JumpStart configuration in the following ways:

- You can add a user-written `finish` script as the following file name:

  jumpstart-dir`/autoscinstall.d/nodes/`*nodename*`/finish`

  The `scinstall` command runs the user-written `finish` scripts after it runs the `finish` script supplied with the product.
- If the directory

  jumpstart-dir`/autoscinstall.d/nodes/`*nodename*`/archive`

  exists, the `scinstall` command copies all files in that directory to the new installation. In addition, if an `etc/inet/hosts` file exists in that directory, `scinstall` uses the hosts information found in that file to supply name-to-address mappings when a name service (NIS/NIS+/DNS) is not used.
- If the directory

  jumpstart-dir`/autoscinstall.d/nodes/`*nodename*`/patches`

exists, the `scinstall` command installs all files in that directory by using the `patchadd`(1M) command. This directory is intended for Solaris software patches and any other patches that must be installed before Sun Cluster software is installed.

You can create these files and directories individually or as links to other files or directories that exist under *jumpstart-dir*.

See the `add_install_client`(1M) man page and related JumpStart documentation for more information about how to set up custom JumpStart install clients.

Run this form of the command from the *install-dir* (see the `-a` form of `scinstall`) on the JumpStart server that you use to initialize the cluster nodes.

Before you use the `scinstall` command to set up a node as a custom Sun Cluster JumpStart client, you must first establish each node as a Solaris install client. The JumpStart directory you specify with the `-c` option to the `add_install_client` command should be the same directory you specify with the `-c` option to `scinstall`. However, the `scinstall` *jumpstart-dir* does not have a server component to it, since you must run the `scinstall` command from a Solaris JumpStart server.

To remove a node as a custom Sun Cluster JumpStart client, simply remove it from the `rules` file.

`-i`

Specifies the "install" form of the `scinstall` command. This form of the command can both install Sun Cluster software and initialize a node as a new cluster member. The new node is the node from which you issue the `scinstall` command.

If the `-F` option is used with `-i`, `scinstall` establishes the node as the first node in a new cluster.

If the `-o` option is used with the `-F` option, `scinstall` establishes a single-node cluster.

If the `-N` option is used with `-i`, `scinstall` adds the node to an already-existing cluster.

If the `-s` option is used and the node is an already-established cluster member, only the specified *srvc* (data service) is installed.

`-p`

Prints release and package versioning information for the Sun Cluster software installed on the node from which the command is run. This is the only form of `scinstall` that you can run as a non-root user.

`-r`

Removes cluster configuration information and uninstall Sun Cluster software from a cluster node. You can then reinstall the node or remove the node from the cluster.

You must run the command on the node that you uninstall, from a directory that is not used by the cluster software, and the node must be in non-cluster mode.

`-u update`
Upgrades Sun Cluster software on the node from which you invoke the `scinstall` command. See `Upgrade Options` below for information specific to the type of upgrade that you intend to perform.

Additional Options

You can combine additional options with the basic options to modify the default behavior of each form of the command. Refer to the `SYNOPSIS` section for additional details about which of these options are legal with which forms of `scinstall`.

The following additional options are supported:

`-d` *cdimage-dir*
Specifies an alternate directory location for finding the media images of the Sun Cluster product and unbundled Sun Cluster data services. The `-d` option is legal with all forms of the command other than the interactive and "print release" (`-p`) forms.

If the `-d` option is not specified, the default directory is the media image from which the current instance of the `scinstall` command is started.

`-h` *nodename*
Specifies the node name. The `-h` option is only legal with the "add install client" (`-c`) form of the command.

The *nodename* is the name of the cluster node (that is, JumpStart install client) to set up for custom JumpStart installation.

`-k`
Specifies that `scinstall` will not install Sun Cluster software packages. The `-k` option is only legal with the "install" (`-i`) form of the command.

If this option is not specified, the default behavior is to install any Sun Cluster packages that are not already installed.

`-s` *srvc*[,…]
Specifies a data service. The `-s` option is only legal with the "install" (`-i`), "upgrade" (`-u`), or "add install client" (`-c`) forms of the command to install or upgrade the specified *srvc* (data service package).

If a data service package cannot be located, a warning message is printed, but installation otherwise continues to completion.

`-v`
Prints release information in verbose mode. The `-v` option is only legal with the "print release" (`-p`) form of the command to specify verbose mode.

In the verbose mode of "print release," the version string for each installed Sun Cluster software package is also printed.

`-F` [*config-options*]
Establishes the first node in the cluster. The `-F` option is only legal with the "install" (`-i`), "upgrade" (`-u`), or "add install client" (`-c`) forms of the command.

The installation of secondary nodes will be blocked until the first node is fully installed, instantiated as a cluster member, and prepared to perform all necessary tasks associated with adding new cluster nodes. If the `-F` option is used with the `-o` option, a single-node cluster is installed and no additional nodes can be added during the installation process.

`-N` *cluster-member* [*config-options*]
Specifies the cluster member. The `-N` option is only legal with the "install" (`-i`), "add install client" (`-c`), "remove" (`-r`), or "upgrade" (`-u`) forms of the command.

- When used with the `-i`, `-c`, or `-u` option, the `-N` option is used to add additional nodes to an existing cluster. The given *cluster-member* is typically the name of the first cluster node established for the cluster. However, it can be the name of any cluster node already participating as a cluster member. The node being initialized is added to the cluster of which *cluster-member* is already an active member. The process of adding a new node to an existing cluster involves updating the configuration data on the given *cluster-member*, as well as creating a copy of the configuration database onto the local file system of the new node.
- When used with the `-r` option, the `-N` option specifies the *cluster-member*, which can be any other node in the cluster that is an active cluster member. The `scinstall` command contacts the specified *cluster-member* to make updates to the cluster configuration. If the `-N` option is not given, `scinstall` makes a best attempt to find an existing node to contact.

Configuration Options

The *config-options* which can be used with the `-F` option or `-N` *cluster-member* option are as follows.

```
media-mnt-pt/Solaris_arch/Product/sun_cluster/Solaris_ver/Tools/scinstall
{-i |  -c  jumpstart-dir  -h  nodename}
[-F
    [ -C  clustername]
    [-G {special | mount-point} ]
    [-T  authentication-options]
    [-A  adapter-options]
    [-B  junction-options]
    [-m  endpoint=[this-node]:name[@port],endpoint=[node:]name[@port]  ]
    [-o]
    [ -w   netaddr-options]
]

media-mnt-pt/Solaris_arch/Product/sun_cluster/Solaris_ver/Tools/scinstall
{-i |  -c  jumpstart-dir  -h  nodename}
[-N  cluster-member
    [-C  clustername]
    [-G {special | mount-point} ]
    [-A  adapter-options]
    [-B  junction-options]
    [-m  endpoint=cable-options]
]
```

`-m` *cable-options*

Specifies the cluster interconnect connections. This option is only legal when the `-F` or `-N` option is also given.

The `-m` option helps to establish the cluster interconnect topology by configuring the cables connecting the various ports found on the cluster transport adapters and junctions. Each new cable configured with this form of the command establishes a connection from a cluster transport adapter on the current node to either a port on a cluster transport junction or an adapter on another node already in the cluster.

If you specify no `-m` options, the `scinstall` command attempts to configure a default cable. However, if you configure more than one transport adapter or junction with a given instance of `scinstall`, it is not possible for `scinstall` to construct a default. The default is to configure a cable from the singly-configured transport adapter to the singly-configured (or default) transport junction.

The `-m` *cable-options* are as follows.

```
-m endpoint=[this-node]:name[@port],endpoint=[node:]name[@port]
```

You must always specify two `endpoint` options with each occurrence of the `-m` option. The *name* component of the option argument specifies the name of either a cluster transport adapter or a cluster transport junction at one of the endpoints of a cable.

- If you specify the *node* component, the *name* is the name of a transport adapter.
- If you do not specify the *node* component, the *name* is the name of a transport junction.

If you specify no *port* component, the `scinstall` command attempts to assume a default port name. The default *port* for an adapter is always `0`. The default port *name* for a junction endpoint is equal to the node ID of the node being added to the cluster.

Refer to the individual cluster transport adapter and cluster transport junction man pages for more information regarding *port* assignments and other requirements. The man pages for cluster transport adapters use the naming convention `scconf_transp_adap_`*adapter*(1M). The man pages for cluster transport junctions use the naming convention `scconf_transp_jct_`*switch*(1M).

Before you can configure a cable, you must first configure the adapters and/or junctions at each of the two endpoints of the cable (see `-A` and `-B`).

The first line in the synopsis given at the beginning of this subsection attempts to express that at least one of the two endpoints must be an adapter on the node being installed. And so, it is not necessary to include *this-node* explicitly. The following is an example of adding a cable:

```
-m endpoint=:hme1,endpoint=switch1
```

In this example, port 0 of the `hme1` transport adapter on this node (the node that `scinstall` is installing) is cabled to a port on transport junction `switch1`. The port used on `switch1` defaults to the node number of this node.

`-o`
Specifies installation and configuration of a single node cluster. This option is only legal when the `-i` and `-F` options are also given.

Other `-F` options are supported, but not required. If the cluster name is not given, the name of the node is used as the cluster name. Transport configuration options may be given, and will be stored in the CCR. The `-G` option is only required if the global devices file system is not the default (/globaldevices). Once a single-node cluster is installed, it is not necessary to configure a quorum device or to disable `installmode`.

`-w` *netaddr-options*
Specifies the private network address. This option is only legal when the `-F` option is also given.

Use this option to specify a private network address (`networks`(4) and, optionally, `netmasks`(4)) for use on the private network. You should only need to use this option when the default private network address collides with an address already in use within the enterprise. The default network address is `172.16.0.0`, with a default netmask of `255.255.0.0`.

The `-w` *netaddr-options* are as follows:

`-w netaddr=`*netaddr*`[,netmask=`*netmask*`]`

`netaddr=`*netaddr*
Specifies the private network address. The default *netaddr* for the private interconnect, or cluster transport, is `172.16.0.0`. The last two octets of this address must always be zero.

[`netmask=`*netmask*]
Specifies the netmask. The default *netmask* for the private interconnect is `255.255.0.0`. The last two octets of the netmask must always be zero, and there cannot be any holes in the mask.

`-A` *adapter-options*
Specifies the transport adapter and, optionally, its transport type. This option is only legal when the `-F` or `-N` option is also given.

Each occurrence of the `-A` option configures a cluster transport adapter attached to the node from which `scinstall` is run.

If no `-A` options are given, an attempt is made to use a default adapter and transport type. The default transport type is `dlpi`. In Sun Cluster 3.1 for SPARC, the default adapter is `hme1`.

When the adapter transport type is `dlpi`, you do not need to specify the `trtype` suboption. In this case, you can use either of the following two forms to specify the `-A` *adapter-options*:

```
-A [trtype=type,]name=adaptername[,vlanid=vlanid][,other-options]
-A adaptername
```

[trtype=*type*]
Specifies the transport type of the adapter. Use the `trtype` option with each occurrence of the `-A` option for which you want to specify the transport type of the adapter. An example of a transport `type` is `dlpi` (see the `sctransp_dlpi`(7P) man pages).

The default transport type is `dlpi`.

name=*adaptername*
Specifies the adapter name. You must use the `name` subobtion with each occurrence of the `-A` option to specify the *adaptername*. An *adaptername* is constructed from a *device name*, immediately followed by a *physical-unit* number (for instance, `hme0`).

If you specify no other suboptions with the `-A` option, you can specify the *adaptername* as a standalone argument to the `-A` option (that is, `-A` *adaptername*).

vlanid=*vlanid*
Specifies the VLAN ID of the tagged-VLAN adapter.

[*other-options*]
Specifies additional adapter options. When a particular adapter provides any other options, you can specify them by using the `-A` option. Refer to the individual man page for the cluster transport adapter for information on any special options that you might use with the adapter.

`-B` *junction-options*
Specifies the transport junction. This option is only legal when the `-F` or `-N` option is also given.

Each occurrence of the `-B` option configures a cluster transport junction. Examples of such devices can include, but are not limited to, Ethernet switches, other switches of various types, and rings.

If you specify no `-B` options, `scinstall` attempts to add a default junction at the time that the first node is instantiated as a cluster node. When you add additional nodes to the cluster, no additional junctions are added by default. However, you can add them explicitly. The default junction is named `switch1`, and it is of type `switch`.

When the junction type is type `switch`, you do not need to specify the `type` suboption. In this case, you can use either of the following two forms to specify the `-B` *junction-options*.

```
-B [type=type,]name=name[,other-options]
-B name
```

If a cluster transport junction is already configured for the given junction *name*, `scinstall` prints a message and ignores the `-B` option.

If you use directly-cabled transport adapters, you are not required to configure any transport junctions. To avoid configuring default transport junctions, use the following special `-B` option:

```
-B type=direct
```

[type=*type*]
Specifies the transport junction type. You can use the `type` option with each occurrence of the `-B` option. Ethernet switches are an example of a cluster transport junction which is of the junction type `switch`. See the individual man page for the cluster transport junction for more information.

You can specify the `type` suboption as `direct` to suppress the configuration of any default junctions. Junctions do not exist in a transport configuration made up of only directly-connected transport adapters. When the `type` suboption is set to `direct`, you do not need to use the `name` suboption.

name=*name*
Specifies the transport junction name. Unless the `type` is `direct`, you must use the `name` subobtion with each occurrence of the `-B` option to specify the transport junction *name*. The *name* can be up to 256 characters in length and is made up of either letters or digits, with the first character being a letter. Each transport junction name must be unique across the namespace of the cluster.

If no other suboptions are needed with `-B`, you can give the junction *name* as a standalone argument to `-B` (that is, `-B` *name*).

[*other-options*]
Specifies additional transport junction options. When a particular junction type provides other options, you can specify them with the `-B` option. Refer to the individual man page for the cluster transport junction for information on any special options that you might use with them.

`-C` *clustername*
Specifies the name of the cluster. This option is only legal when the `-F` or `-N` option is also given.

- If the node being installed is the first node in a new cluster, the default *clustername* is the same as the name of the node being installed (or when upgrading, if it exists, the current cluster's *clustername* will be used as the default *clustername*).
- If the node being installed is being added to an already-existing cluster, the default *clustername* is the name of the cluster to which *cluster-member* already belongs.

It is an error to specify a *clustername* that is not the name of the cluster to which *cluster-member* belongs.

`-G` {*special* | *mount-point*}
Specifies a raw *special* disk device or a file system for the global-devices mount point. This option is only legal when the `-F`, `-N`, or `-r` option is also given.

- When used with the `-F` or `-N` option, the `-G` option specifies the raw *special* disk device or the file system *mount-point* to use in place of the `/globaldevices` mount point. Each cluster node must have a local file system mounted globally on `/global/.devices/node@`*nodeID* before the node can successfully participate as a cluster member. However, since the node ID is not known until the `scinstall` command is run, `scinstall` attempts to add the necessary entry to the `vfstab`(4) file when it does not find a `/global/.devices/node@`*nodeID* mount.

  By default, the `scinstall` command looks for an empty file system mounted on `/globaldevices`. If such a file system is provided, the `scinstall` command makes the necessary changes to the `vfstab` file. These changes create a new `/global/.devices/node@`*nodeID* mount point and remove the default `/globaldevices` mount point. However, if `/global/.devices/node@`*nodeID* is not mounted and an empty `/globaldevices` file system is not provided, the `-G` option must be given to specify the raw *special* disk device or the file system *mount-point* to use in place of `/globaldevices`.

  If a raw *special* disk device name is given and `/global/.devices/node@`*nodeID* is not mounted, a file system is created on the device using the `newfs`(1M) command. It is an error to supply the name of a device with an already-mounted file system.

  As a guideline, this file system should be at least 512 Mbytes in size. If this partition or file system is not available, or is not large enough, it might be necessary to reinstall the Solaris operating environment.
- When used with the `-r` option, the `-G` *mount-point* option specifies the new mount-point name to use to restore the former `/global/.devices` mount point. If the `-G` option is not specified, the mount point is renamed `/globaldevices` by default.

`-T` *authentication-options*
Specifies node-authentication options for the cluster. This option is only legal when the `-F` option is also given.

Use this option to establish authentication policies for nodes that attempt to add themselves to the cluster configuration. Specifically, when a machine requests that it be added to the cluster as a cluster node, a check is made to determine whether or not the node has permission to join. If the joining node has permission, it is authenticated and allowed to join the cluster.

You can only use the `-T` option with the `scinstall` command when you set up the very first node in the cluster. If the authentication list or policy needs to be changed on an already-established cluster, use the `scconf`(1M) command.

The default is to allow any machine to add itself to the cluster.

The `-T` *authentication-options* are as follows.

`-T node=`*nodename*`[,...][,authtype=`*authtype*`]`

node=*nodename*[,…]
Specifies node names to add to the node authentication list. You must specify at least one `node` suboption to the `-T` option. This option is used to add node names to the list of nodes that are able to install and configure themselves as nodes in the cluster. If the authentication list is empty, any node can request that it be added to the cluster configuration. However, if the list has at least one name in it, all such requests are authenticated using the authentication list. You can modify or clear this list of nodes at any time by using the `scconf`(1M) command from one of the active cluster nodes.

[`authtype`=*authtype*]
Specifies the type of node authentication. The only currently-supported `authtypes` are `des` and `sys` (or, `unix`). If no `authtype` is specified, `sys` is the default.

If you will you specify `des` (Diffie-Hellman) authentication, first add entries to the `publickey`(4) database for each cluster node to be added before you run the `-T` option to the `scinstall`command.

You can change the authentication type at any time by using the `scconf`(1M) command from one of the active cluster nodes.

Patch Options

The `-M` option installs the patches in the patch directory during the `scinstall` process by using the `patchadd`(1M) command. The *patch-options* to `-M` are as follows.

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`
`[-M patchdir=`*dirname*`[,patchlistfile=`*filename*`]]`

Note – If you use the `-M` option, the `scinstall` command ignores the patch directory inside the *jumpstart-dir* directory.

patchdir=*dirname*
Specifies the path to the directory that contains the patches required for Sun Cluster. This directory must be on a file system that is accessible by all nodes.

If you are including Solaris patches in the `/var/cluster/patches` directory, view the `/etc/release` file to see the exact version of Solaris software that is installed on a node.

patchlistfile=*filename*
Specifies a file containing the list of patches to install. If you do not specify a patch list file, the `scinstall` command will install all the patches in the *dirname* directory, including tarred, jarred, and zipped patches.

For information on creating a patch list file, refer to the `patchadd`(1M) manual page.

Upgrade Options

Use the `-u update` option to upgrade Sun Cluster software. The *upgrade-options* to `-u update` are as follows.

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/Tools/scinstall`
`-u update [-s {`*srvc*`[,...] | all}] [-d` *cdimage-dir*`] [ -O ]`
`[-S { interact | testaddr=`*testipaddr@adapter*`[,testaddr=...]} ]`

`-s {`*srvc*`[,...] | all}`
Upgrades data services. If the `-s` option is not specified, only cluster framework software is upgraded. If the `-s` option is specified, only the specified data services are upgraded.

The following suboption to the `-s` option is specific to the `update` mode of upgrade:

`all` Upgrades all data services.

This suboption to `-s` is only legal with the `update` mode.

This suboption upgrades all data services currently installed on the node, except those data services for which an update version does not exist in the update release.

The `-s` option is not compatible with the `-S` test IP address option.

`-O`
Overrides the hardware validation and bypasses the version-compatibility checks.

`-S {interact | testaddr=`*testipaddr@adapter*`[,testaddr=...]`
Specifies test IP addresses. This option allows the user either to direct the command to prompt the user for the required IP Network Multipathing addresses or to supply a set of IP Network Multipathing test addresses on the command line for the conversion of NAFO to IP Network Multipathing groups. See "Introducing IPMP (Overview)" in *System Administration Guide: IP Services* for additional information on IP Network Multipathing.

Note – The `-S` option is only required when one or more of the NAFO adapters in `pnmconfig` is not already converted to use IP Network Multipathing.

The suboptions of the `-S` option are the following:

`interact`
Prompt the user to supply one or more IP Network Multipathing test addresses individually.

`testaddr=`*testipaddr@adapter*
Allow the user to specify one or more IP Network Multipathing test addresses without being prompted for the list.

testipaddr
The IP address or hostname (in the `/etc/inet/hosts` file) that will be assigned as routable, no-failover and deprecated test IP address to the adapter. IP Network Multipathing uses test addresses to detect failures and repairs. See "IPMP Addressing" in *System Administration Guide: IP Services* for additional information on configuring test IP addresses.

adapter
The name of the NAFO network adapter to be added to an IP Network Multipathing group.

It is illegal to combine both the `interact` and the `testaddr` suboptions on the same command line.

EXAMPLES

Installing and Initializing a Two-Node Cluster

The following sequence of commands installs and initializes a typical two-node cluster with Sun Cluster software for Solaris 9 on SPARC.

Insert the framework installation media on `node1` and issue the following commands:

```
node1# cd media-mnt-pt/Solaris_sparc/Product/sun_cluster/Solaris_9/Tools/
node1# ./scinstall -i -F
```

Insert the framework installation media on `node2` and issue the following commands:

```
node2# cd media-mnt-pt/Solaris_sparc/Product/sun_cluster/Solaris_9/Tools/
node2# ./scinstall -i -N node1
```

Installing and Initializing a Single-Node Cluster

The following commands install and initialize a single-node cluster with Sun Cluster software for Solaris 9 on SPARC, with all defaults accepted. Insert the framework installation media and issue the following commands:

```
# cd media-mnt-pt/Solaris_sparc/Product/sun_cluster/Solaris_9/Tools/
# ./scinstall -i -F -o
```

Setting Up a Solaris Installation Server

The following sequence of commands arranges to set up a Solaris installation server to install and initialize Sun Cluster software, for Solaris 9 on SPARC, on a three-node SCI–PCI cluster. Insert the framework installation media on the installation server and issue the following commands:

```
# cd media-mnt-pt/Solaris_sparc/Product/sun_cluster/Solaris_9/Tools/
# ./scinstall -a /export/sc3.1
# cd /export/sc3.1/Solaris_sparc/Product/sun_cluster/Solaris_9/Tools/
# ./scinstall -c /export/jumpstart -h node1 -F -A hme2
# ./scinstall -c /export/jumpstart -h node2 -N node1 -A hme2
# ./scinstall -c /export/jumpstart -h node3 -N node1 -A hme2
```

Upgrading the Framework and Data Service Software

The following sequence of commands upgrades the framework and data service software of a cluster to the next Sun Cluster release. The first example uses the Sun Cluster version for Solaris 9 on SPARC and the second example uses the Sun Cluster version for Solaris 9 on x86. Carry out these operations on each cluster node. Insert the framework installation media and issue the following commands:

SPARC

```
ok> boot -x
# cd media-mnt-pt/Solaris_sparc/Product/sun_cluster/Solaris_9/Tools/
# ./scinstall -u update -S interact
# cd /
# eject cdrom
```

Insert the Agents installation media and issue the following commands:

```
# /usr/cluster/bin/scinstall -u update -s all -d /cdrom/cdrom0
# reboot
```

x86

```
                     <<< Current Boot Parameters >>>
Boot path: /pci@1,0/pci8086,340f@7,1/sd@0,0:a
Boot args:

Type    b [file-name] [boot-flags] <ENTER>      to boot with options
or      i <ENTER>                               to enter boot interpreter
or      <ENTER>                                 to boot with defaults

                 <<< timeout in 5 seconds >>>

Select (b)oot or (i)nterpreter: b -x
...
# cd media-mnt-pt/Solaris_x86/Product/sun_cluster/Solaris_9/Tools/
# ./scinstall -u update -S interact
# cd /
# eject
```

Insert the Agents installation media and issue the following commands:

```
# /usr/cluster/bin/scinstall -u update -s all -d /cdrom/cdrom0
# reboot
```

Uninstalling a Node

The following sequence of commands places the node in non-cluster mode, then removes Sun Cluster software and configuration information from the cluster node, renames the global-devices mount point to the default name `/globaldevices`, and performs cleanup:

SPARC

```
ok> boot -x
# cd /
# /usr/cluster/bin/scinstall -r
```

x86

```
                     <<< Current Boot Parameters >>>
Boot path: /pci@1,0/pci8086,340f@7,1/sd@0,0:a
Boot args:

Type    b [file-name] [boot-flags] <ENTER>      to boot with options
or      i <ENTER>                               to enter boot interpreter
or      <ENTER>                                 to boot with defaults

                 <<< timeout in 5 seconds >>>

Select (b)oot or (i)nterpreter: b -x
...
# cd /
# /usr/cluster/bin/scinstall -r
```

EXIT STATUS

The following exit values are returned:

`0`	Successful completion.
`non-zero`	An error occurred.

FILES

media-mnt-pt`/.cdtoc`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/.producttoc`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/ \`
`  Packages/.clustertoc`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/ \`
`  Packages/.order`

media-mnt-pt`/Solaris_`*arch*`/Product/sun_cluster/Solaris_`*ver*`/ \`
`  Tools/defaults`

media-mnt-pt`/components/`*srvc*`/Solaris_`*ver*`/Packages/.clustertoc`

media-mnt-pt`/components/`*srvc*`/Solaris_`*ver*`/Packages/.order`

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	Sun Cluster installation media, Java Enterprise Systems installation media, SUNWscu
Interface Stability	Evolving

SEE ALSO

`add_install_client`(1M), `newfs`(1M), `patchadd`(1M), `sccheck`(1M), `scconf`(1M), `scconf_transp_adap_bge`(1M), `scconf_transp_adap_eri`(1M), `scconf_transp_adap_ge`(1M), `scconf_transp_adap_hme`(1M), `scconf_transp_adap_ibd`(1M), `scconf_transp_adap_qfe`(1M), `scconf_transp_adap_sci`(1M), `scconf_transp_jct_dolphinswitch`(1M), `scconf_transp_jct_etherswitch`(1M), `scconf_transp_jct_ibswitch`(1M), `setup_install_server`(1M), `clustertoc`(4), `netmasks`(4), `networks`(4), `order`(4), `packagetoc`(4), `sctransp_dlpi`(7P)

Sun Cluster Software Installation Guide for Solaris OS

System Administration Guide: IP Services

NAME

scnas – manage network-attached storage (NAS) device configuration data for Sun Cluster.

SYNOPSIS

scnas -a [-H] [-n] -h *device-name* -t *device-type* -o *specific-options* [-f *input-file*]

scnas -c [-H] [-n] -h *device-name* -o *specific-options* [-f *input-file*]

scnas -r [-H] -h *device-name*

scnas -p [-H] [-h *device-name*] [-t *device-type*]

scnas [-H]

DESCRIPTION

The scnas command manages NAS devices in a Sun Cluster configuration. To manage NAS directories in the cluster, use the scnasdir(1M) command.

You can use the scnas command to create the NAS device configuration, to update the NAS type-specific properties, and to remove the device configuration from Sun Cluster. The options to this command are processed in the order in which they are typed on the command line.

The scnas command can only be run from an active cluster node. The results of running the command are always the same, regardless of the nodethat is used.

All forms of the scnas command accept the -H option. Specifying -H displays help information. All other options are ignored. Help information is also printed when scnas is run without options.

The NAS device must be set up before using the scnas command to manage a NAS device. Refer to the documentation for the particular NAS device for procedures for setting up a device.

OPTIONS

Basic Options

The following options are common to all forms of the scnas command:

-H
: If this option is specified on the command line at any position, the command prints help information. All other options are ignored and are not executed. Help information is also printed if scnas is run with no options.

-n
: If this option is specified on the command line at any position, the scnas command only checks the usage and does not write the configuration data. If the -n option is specified with the -f option, the scnas command checks the input file for the password.

The following options modify the basic form and function of the scnas command. None of these options can be combined on the same command line.

`-a`
Specifies the `add` form of the `scnas` command. The `-a` option can be used to add a NAS device into the Sun Cluster configuration. Additional associated properties of the device need to be specified.

`-c`
Specifies the `change` form of the `scnas` command. The `-c` option is used to change specific NAS device properties.

`-r`
Specifies the `remove` form of the `scnas` command. The `-r` option is used to remove the NAS device from the Sun Cluster configuration. Before removing a device, all its exported directories must be removed by using `scnasdir`.

`-p`
Specifies the `print` form of the `scnas` command. When no other options are given, the `-p` option prints a listing of all the current NAS devices configured in Sun Cluster and all their associated properties. This option can be used with additional options to query a particular device or a particular type of device.

Additional Options

The following additional options can be combined with one or more of the previously described basic options to configure all properties for a device. The device does not need to be online to use these options. Refer to the SYNOPSIS section to see the options that can be used with each form of `scnas`.

The additional options are as follows:

`-h` *device-name*
Use this option to specify the name of the NAS device in the Sun Cluster configuration. The device name identifies the device and can be used to remotely access the device by using `rhs` or `telnet`.

This device name must be specified for the `add`, `change`, and `remove` forms of the `scnas` command.

`-t` *device-type*
The NAS device type. Currently, the NAS device type is identified by the vendor name.

For example, the NAS device type for Network Appliance, Inc. is `netapp`. This option is required when you add a NAS device to the Sun Cluster configuration.

`-o` *specific-options*
Use this option to provide the properties that are specific to a NAS device type. For example, the NAS device from Network Appliance, Inc. has the following property:

`-o userid=`*`userid`*

The `userid` property is used by the cluster to perform administrative duties on the device. When you add a `userid` to the device configuration, you are prompted for its password. You can also place the password in a text file and use it by specifying the `-f` option.

`-f` *input-file*

For security reasons, the password cannot be specified in command-line options. To keep the password secure, place it in a text file and specify the file by using the `-f` option. If you do not specify an input file for the password, the command prompts for the password.

Set permissions of the input file to readable by root and prohibit access by either group or world.

In the input file, the password cannot be entered across multiple lines. Leading white spaces and tabs are ignored. Comments begin with an unquoted # sign, and continue to the next new line.

The parser ignores all comments. When you use an input file for the device user password, the # sign cannot be part of the password.

EXAMPLES

EXAMPLE 1 Adding a NAS Device to a Cluster

The following `scnas` command adds a Network Appliance, Inc. storage system to the Sun Cluster configuration.

```
# scnas -a -h netapp1 -t netapp -o userid=root
Please enter password:
```

EXAMPLE 2 Removing a NAS Device From a Cluster

The following `scnas` command removes a NAS device from the Sun Cluster configuration.

```
# scnas -r -h netapp1
```

EXIT STATUS

The following exit values are returned:

`0`	The command executed successfully.
nonzero	An error has occurred.

SEE ALSO

`scconf`(1M), `scnasdir`(1M)

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Stability	Evolving

NAME

scnasdir – manage the exported directories on a network-attached storage (NAS) device in a Sun Cluster configuration.

SYNOPSIS

```
scnasdir [-a] [-H] [-n] -h device-name [-d directory [-d directory...]]
     [-f input-file]

scnasdir -r [-H ] [-n ] -h device-name [-d all | -d directory
     [-d directory...]] [-f input-file]

scnasdir -p [-H] [-h device-name] [-t device-type]

scnasdir [-H]
```

DESCRIPTION

The `scnasdir` command manages the exported directories on NAS devices in a Sun Cluster configuration. The device must already have been configured in the cluster by using the `scnas` command.

The `scnasdir` command can be used to add directories to a device's cluster configuration, to remove directories from a device's cluster configuration, and to print the directories of a particular device or particular device types.

The options in this command are processed in the order in which they are typed on the command line. The `scnasdir` command can only be run from an active cluster node. The results of running the command are always the same, regardless of the node that is used.

All forms of the `scnasdir` command accept the `-H` option. Specifying `-H` displays help information, and all other options are ignored and not executed. Help information is also printed when `scnasdir` is run without options.

OPTIONS

Basic Options

The following options are common to all forms of the `scnasdir` command:

`-H`
: If this option is specified on the command line at any position, the command prints help information. All other options are ignored and are not executed. Help information is also printed if `scnasdir` is run with no options.

`-n`
: If this option is specified on the command line at any position, the `scnasdir` command only checks the usage and does not write the configuration data. If the `-n` option is specified with the `-f` option, the `scnasdir` command displays the data that will be processed for the user to review.

The following options modify the basic form and function of the `scnasdir` command. None of these options can be combined on the same command line.

`-a`
: Specifies the `add` form of the `scnasdir` command. The `-a` option can be used to add directories into the device's Sun Cluster configuration.

`-r`
Specifies the `remove` form of the `scnasdir` command. The `-r` option is used to remove all the directories, or the specified directories of a NAS device from its Sun Cluster configuration.

`-p`
Specifies the `print` form of the `scnasdir` command. When no other option is given, this `-p` option prints a listing of all the directories of all the NAS devices configured in Sun Cluster. This option can be used with additional options to query a particular device or particular types of NAS devices.

Additional Options

The following additional options can be combined with one or more of the previously described basic options to manage the directories of a device.

The additional options are as follows:

`-h` *device-name*
Use this option to specify the name of the NAS device in the Sun Cluster configuration. The `-h` option identifies the device and can be used to remotely access the device by using `rhs` or `telnet`.

This device name must be specified for the `add`, `change`, and `remove` forms of the `scnasdir` command.

`-d` *all* | *directory*
Use this option to list the directories (or volumes) exported on the NAS device to be configured into the Sun Cluster. These directories must be created and exported on the device before using `scnasdir`. See the documentation for the NAS device type for procedures for exporting directories.

The `-d all` option can only be accepted by the remove format, `-r`.

The directories must be specified by using either the `-d` option, or the `-f` option, for the `add` and `remove` forms of the `scnasdir` command.

`-f` *input-file*
Directories can be placed into a plain text file, one directory per line, and used with the `-f` option. Leading white spaces and tabs are ignored. Comments begin with an unquoted # sign, and continue to the next new line. The parser ignores all comments.

EXAMPLES

EXAMPLE 1 Adding Two NAS Storage Device Directories to a Cluster

The following `scnasdir` command adds two directories of a NAS device to the Sun Cluster configuration.

```
# scnasdir -a -h netapp1 -d /vol/DB1 -d /vol/DB2
```

EXAMPLE 2 Removing All of a NAS Storage Device's Directories From a Cluster

The following `scnasdir` command removes all the directories that are configured for a NAS device.

EXAMPLE 2 Removing All of a NAS Storage Device's Directories From a Cluster *(Continued)*

```
# scnasdir -r -h netapp1 -d all
```

EXIT STATUS

The following exit values are returned:

0	The command executed successfully.
nonzero	An error has occurred.

SEE ALSO

scconf(1M), scnas(1M)

attributes(5)

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

NAME

scrgadm – manage registration and unregistration of resource types, resource groups, and resources

SYNOPSIS

Show Current Configuration

scrgadm -p[v[v]] [-t *resource_type_name*] [-g *resource_group_name*]
[-j *resource_name*]

Resource Type Commands

scrgadm -a -t *resource_type_name* [-h *RT_installed_node_list*]
[-f *registration_file_path*]

scrgadm -c -t *resource_type_name* [-h *RT_installed_node_list*]
[-y RT_system={TRUE|FALSE}]

scrgadm -r -t *resource_type_name*

Resource Group Commands

scrgadm -a -g *RG_name* [-h *nodelist*] [-y *property*...]

scrgadm -c -g *RG_name* [-h *nodelist*] -y *property*...

scrgadm -r -g *RG_name*

Resource Commands

scrgadm -a -j *resource_name* -t *resource_type_name* -g *RG_name* [-y *property*...]
[-x *extension_property*...]

scrgadm -c -j *resource_name* [-y *property*...] [-x *extension_property*...]

scrgadm -r -j *resource_name*

Logical Host Name Resource Commands

scrgadm -a -L -g *RG_name* -l *hostnamelist* [-j *resource_name*] [-n *netiflist*]
[-y *property*...]

Shared Address Resource Commands

scrgadm -a -S -g *RG_name* -l *hostnamelist* [-j *resource_name*] [-n *netiflist*]
[-X *auxnodelist*] [-y *property*...]

DESCRIPTION

A resource type specifies common properties and callback methods for all resources of that type. Before you can create a resource of a particular type, you must first register the resource type using the following form of the command:

scrgadm -a -t *resource_type_name*

A resource group contains a set of resources, all of which are brought online or offline together on a given node or set of nodes. You first create an empty resource group before placing any resources in it. To create a resource group, use the following command:

scrgadm -a -g *RG_name*

There are two types of resource groups: failover and scalable.

A failover resource group is online on only one node at a time. A failover resource group can contain resources of any type although scalable resources that are configured in a failover resource group run on only one node at a time.

To create a failover resource group named `MyDatabaseRG`, use the following command:

```
# scrgadm -a -g MyDatabaseRG
```

A scalable resource group can be online on several nodes at once. A scalable resource group can contain only resources that support scaling and cannot contain resources that are constrained, by their resource type definition, to only failover behavior.

To create a scalable resource group named `MyWebServerRG`, use the following command:

```
# scrgadm -a -g MyWebServerRG \
-y Maximum_primaries=integer \
-y Desired_primaries=integer
```

A newly created resource group is in an `UNMANAGED` state. After creating resources in the group, use the `scswitch`(1M) command to put a resource group in a `MANAGED` state.

To create a resource of a given type in a resource group, use the following command:

```
# scrgadm -a -j resource_name -t resource_type_name -g RG_name
```

Creating a resource causes the underlying RGM mechanism to take several actions. The underlying RGM mechanism calls the `VALIDATE` method on the resource to verify that the property settings of the resource are valid. If the `VALIDATE` method completes successfully and the resource group has been put in a `MANAGED` state, the RGM initializes the resource by calling the `INIT` method on the resource. The RGM then brings the resource online if it is enabled and its resource group is online.

To remove a resource group, first remove all resources from that resource group. To remove a resource, first disable it with the `scswitch`(1M) command. Removing a `resource` causes the RGM to clean up after the resource by calling the `FINI` method on that resource.

OPTIONS

Action Options

Action options specify the actions performed by the command. Only one action option is allowed on the command line.

The following action options are supported:

`-a`
Adds a new configuration. Use with these options:

`-g` Creates a resource group.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-j` Creates a resource.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-t` Adds a resource type.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-c`

Modifies an existing configuration. Only values of the specified properties are set. Other properties retain their current values. Use with these options:

`-g` Modifies a resource group.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-j` Modifies a resource.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-t` Modifies a resource type.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-r`

Removes configuration. Use with these options:

`-g` Removes a resource group.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-j` Removes a resource.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-t` Removes a resource type.

You need `solaris.cluster.resource.modify` RBAC authorization to use this command option with `-a`, `-c`, or `-r`. See `rbac`(5).

`-p`

Displays existing configuration information. Use with these options:

`-g` *resource_group_name*
Displays specific resource group configuration information.

You need `solaris.cluster.resource.read` RBAC authorization to use this command option with `-p`. See `rbac`(5).

`-j` *resource_name*
Displays specific resource configuration information.

You need `solaris.cluster.resource.read` RBAC authorization to use this command option with `-p`. See `rbac`(5).

`-t` *resource_type_name*
Displays specific resource type configuration information.

You need `solaris.cluster.resource.read` RBAC authorization to use this command option with `-p`. See `rbac`(5).

`-v`[`v`]
Displays more verbose output.

You need `solaris.cluster.resource.read` RBAC authorization to use this command option with `-p`. See `rbac`(5).

If you do not specify any `-g`, `-j`, or `-t` options, information about all resource types, resource groups, and resources that are currently configured on the cluster are provided by default.

Multiple `-g`, `-j`, and `-t` options are supported and can be combined with any combination of `-v` options.

You can use up to two `-v` options on a single command line.

Target Options

Target options identify the target object. The following target options are supported:

Note – Property names for resource groups, resources, and resource types are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify property names.

`-g` *RG_name*
Resource group.

`-j` *resource_name*
Resource. When used with the `-a` option, the `-t` and `-g` target options must be specified in the command to indicate the type of the resource that is to be instantiated and the name of the containing resource group.

`-t` *resource_type_name*
Resource type.

Resource Type-Specific Options

The following options are supported:

`-f` *registration_file_path*
Is valid with `-a`. Specifies the path name of the resource type registration file and is required if the file is not in the well-known directory (usually `/usr/cluster/lib/rgm/rtreg`).

`-h` *RT_installed_node_list*
Is valid with `-a` and `-c`. Specifies a comma-separated list of node names upon which this resource type is installed. Resources of this type can be instantiated only in resource groups whose nodelist is a subset of this list.

`-h` is optional with the `-a` option. If `-h` is not specified, it implies that the resource type has been installed on all nodes. Doing so permits resources of this type to be instantiated in any resource group.

When used with the `-c` option, `-h` must be specified with either a new installed node list or with an escaped wildcard character (`\*`). The wildcard character indicates that the resource type has been installed on all nodes.

Note – A comma is not allowed in a node name.

`-t` *resource_type_name*
Is valid with `-a`, `-c`, and `-r`. A resource type is defined by a resource type registration file that specifies standard and extension property values for the resource type. Placing a valid resource type registration file in the well-known directory where registration files are usually installed (`/usr/cluster/lib/rgm/rtreg`) enables the shorthand notation:

```
# scrgadm -a -t SUNW.rt:2.0
```

As a result, you do not need to use the following notation:

```
# scrgadm -a -t rtn -f full_path_to_SUNW.rt:2.0
```

To view the names of the currently registered resource types, use the following command:

```
# scrgadm -p
```

Starting in Sun Cluster 3.1, the syntax of a resource type name is as follows:

vendor_id`.`*resource_type*`:`*version*

The three components of the resource type name are properties specified in the RTR file as *Vendor_id*, *Resource_type*, and *RT_version*. The `scrgadm` command inserts the period and colon delimiters. The optional *Vendor_id* prefix is necessary only if it is required to distinguish between two registration files of the same name provided by different vendors. The *RT_version* is used for upgrading from one version of a data service to another version of the data service.

To ensure that the *Vendor_id* is unique, use the stock symbol for the company that is creating the resource type. The *resource_type_name* that is used with the `-t` option can either be the full resource type name or an abbreviation that omits the *Vendor_id*. For example, both `-t SUNW.iws` and `-t iws` are valid. If there are two resource types in the cluster with names that differ only in the *Vendor_id* prefix, the use of the abbreviated name fails.

The `scrgadm` command fails to register the resource type if the *RT_version* string includes a blank, tab, slash (`/`), backslash (`\`), asterisk (`*`), question mark (`?`), left square bracket (`[`), or right square bracket (`]`) character.

When you specify the *resource_type_name* with the `-t` option, you can omit the version component if only one version is registered.

Resource type names that you created before the Sun Cluster 3.1 release continue to conform to the following syntax:

vendor_id`.`*resource_type*

`-y RT_system=`{`TRUE`|`FALSE`}
Sets the `RT_system` property of a resource type either to `TRUE` or to `FALSE`. The default value of the `RT_system` property is `FALSE`. See `rt_properties(5)` for a description of the `RT_system` property.

Resource Group-Specific Options

The following options are supported:

`-h` *nodelist*
Is valid with `-a` and `-c`. This option is a shortcut for `-y Nodelist=`*nodelist*.

`-y` *property*
Is valid with `-a` and `-c`. A *property* is defined as a *name*=*value* pair. Multiple instances of `-y` *property* are allowed. The form of the *value* is dictated by each *property*. In the following example, *property1* takes a single string as the *value*, while *property2* takes a comma-separated string array:

```
-y property1=value1 -y property2=value2a,value2b
```

To set a string property to an empty value, use this option without specifying a value, as follows:

```
-y property=
```

Recognition of `-y` property names is not case-sensitive.

See `rg_properties(5)` for a description of the resource group properties.

Resource-Specific Options

The following options are supported:

`-x` *extension_property*
Is valid with `-a` and `-c`. An *extension_property* is defined as a *name*=*value* pair that is applicable only to a given resource type. Multiple instances of `-x` *extension_property* are allowed. The form of *value* is dictated by each *extension_property*. In the following example, *extension_property1* takes a single string as the *value*, while *extension_property2* takes a comma-separated string array:

```
-x extension_property1=value1 \
-x extension_property2=value2a,value2b
```

For information about the extension properties that are available for a particular data service, see the man page for that data service.

`-y` *property*
Is valid with `-a` and `-c`. A *property* is defined as a *name*=*value* pair. Multiple instances of `-y` *property* are allowed. The form of the *value* is dictated by each *property*. In the following example, *property1* takes a single string as the *value*, while *property2* takes a comma-separated string array:

-y *property1*=*value1* **-y** *property2*=*value2a*,*value2b*

To set a property to an empty value, use this option without specifying a value, as follows:

-y *property*=

Recognition of -y *property* names is not case-sensitive.

See the r_properties(5) man page for a description of the resource properties.

LogicalHostName Specific Options

These options apply to logical host name resources. There are no special commands for removing a LogicalHostname resource:

```
# scrgadm -r -j resource_name
```

resource_name is the same name that is supplied with the optional -j option when you create the LogicalHostname resource. If the -j option and *resource_name* are omitted when the LogicalHostname resource is created, then the name is generated by scrgadm.

The following options are supported:

-j *resource_name*
The -j option is required when you use an IP address rather than a host name as the first argument to the -l *hostnamelist* option.

Use -j with -a to explicitly name a LogicalHostname resource when the resource is created and with -r to remove a resource from a resource group. If you do not use the -j option to explicitly name the resource, the scrgadm command creates the resource and assigns the name of the first host name in *hostnamelist* to that resource.

-L
Indicates that the options that are used on the command line apply to a logical host name. If you issue the command when any cluster node is not an active cluster member, you must also use the -n *netiflist* option.

-l *hostnamelist*
Specifies the IPv4 or IPv6 addresses to be shared. Use host names even though you can specify IP addresses. *hostnamelist* is a comma-separated list of host names that are to be made available by this LogicalHostname resource.

-n *netiflist*
Specifies the list of network interfaces. The -L option requires the -n option if the command is issued when any cluster node is not an active cluster member.

The *netiflist* takes the following form:

netif@*node*[,...]

netif may be given as network adapter name, such as le0, or as an IP Network Multipathing group name, such as sc_ipmp. The *node* may be a node name or node identifier. All nodes in the *nodelist* of the resource group must be represented in

netiflist. If `-n` *netiflist* is omitted, an attempt is made to discover a net adapter on the subnet identified by the *hostnamelist* for each node in the *nodelist*. Single-adapter IP Network Multipathing groups are created for discovered network adapters not already in an IP Network Multipathing group. Similarly, a single-adapter IP Network Multipathing group is created for a named adapter, if a group does not already exist.

Refer to the NOTES section for more information.

`-y` *property*
Refer to the Resource-Specific Options section for details.

`SharedAddress` Specific Options

All of the `LogicalHostname`-specific options also apply to `SharedAddress` resources with the following changes and additions:

`-S`
Indicates that the options that are used on the command line apply to a shared address.

`-X` *auxnodelist*
Specifies a comma-separated list of node names or identifiers. Entries on this list must be members of the cluster. These nodes are nodes that may host the specified shared addresses, but never serve as the primary node in the case of failover.

This list is mutually exclusive with *nodelist*. See the description of *nodelist* under Resource Group-Specific Options.

EXIT STATUS

The following exit values are returned:

`0` The command completed successfully.

A warning message might be written to the standard error even when this command completes successfully.

nonzero An error has occurred.

Writes an error message to standard error when it exits with nonzero status.

Some operations are not permitted on resource types whose `RT_System` property is `TRUE`. Similarly, some operations are not permitted on a resource group (and its resources) whose `RG_System` property is `TRUE`. See `rt_properties(5)` and `rg_properties(5)`.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWcsu`
Interface Stability	Evolving

SEE ALSO `ifconfig`(1M), `scstat`(1M), `scswitch`(1M), `attributes`(5), `r_properties`(5), `rbac`(5), `rg_properties`(5), `rt_properties`(5)

NOTES A network adapter that is not already configured for use cannot be discovered or placed into an IP Network Multipathing group during `LogicalHostname` and `SharedAddress` add operations. See `ifconfig`(1M).

If `scrgadm` exits nonzero with the error message `cluster is reconfiguring`, the requested operation might have completed successfully, despite the error status. If you doubt the result, you can execute `scrgadm` again with the same arguments after the reconfiguration is complete.

NAME

scsetup – interactive cluster configuration tool

SYNOPSIS

scsetup [-f *logfilename*]

DESCRIPTION

At post-install time, the scsetup utility performs initial setup tasks, such as configuring quorum devices and resetting *installmode*. Always run the scsetup utility just after the cluster has been installed and all of the nodes have joined for the first time.

Once *installmode* has been disabled, scsetup provides a menu-driven front end to most ongoing cluster administration tasks.

You can execute scsetup from any node in the cluster. However, when installing a cluster for the first time, it is important to wait until all nodes have joined the cluster before running scsetup and resetting *installmode*.

OPTIONS

The following options are supported:

-f *logfilename* Specify the name of a log file to which commands can be logged. If this option is specified, most command sets generated by scsetupcan be run and logged, or just logged, depending on user responses.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes.

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

scconf(1M), scrgadm(1M), scswitch(1M), attributes(5)

NAME

scshutdown – shut down a cluster

SYNOPSIS

scshutdown [-y] [-g *grace-period*] [*message*]

DESCRIPTION

The scshutdown utility shuts down an entire cluster in an orderly fashion.

Before starting the shutdown, scshutdown sends a warning message, then a final message asking for confirmation.

Only run the scshutdown command from one node.

scshutdown performs the following actions when it shuts down a cluster:

- Changes all functioning resource groups on the cluster to an offline state. If any transitions fail, scshutdown does not complete and displays an error message.
- Unmounts all cluster file systems. If any unmounts fail, scshutdown does not complete and displays an error message.
- Shuts down all active device services. If any transition of a device fails, scshutdown does not complete and displays an error message.
- Runs /usr/sbin/init 0 on all nodes. See init(1M) for more information.

You need solaris.cluster.system.admin RBAC authorization to use this command. See rbac(5).

OPTIONS

The following options are supported:

-g *grace-period* — Changes the number of seconds from the 60-second default to the time specified by *grace-period*.

-y — Pre-answers the confirmation question so the command can be run without user intervention.

OPERANDS

The following operands are supported:

message — Is a string that is issued after the standard warning message The system will be shut down in ... is issued. If *message* contains more than one word, delimit it with single (') or double (") quotation marks. The warning message and the user-provided *message* are output when there are 7200, 3600, 1800, 1200, 600, 300, 120, 60, and 30 seconds remaining before scshutdown begins.

EXAMPLES

EXAMPLE 1 Shutting Down a Cluster

```
phys-palindrome-1# scshutdown
```

EXIT STATUS

The following exit values are returned:

0 — The command completed successfully.

nonzero	An error occurred. Error messages are displayed on the standard output.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

`shutdown`(1M), `init`(1M), `attributes`(5)

NAME

scsnapshot – retrieve configuration data about resource groups, resource types, and resources, and generate a shell script

SYNOPSIS

scsnapshot [-s *scriptfile*] [-o *imagefile*]

scsnapshot [-s *scriptfile*] *oldimage newimage*

DESCRIPTION

The scsnapshot tool retrieves information from the Cluster Configuration Repository (CCR) about configuration data that is related to resource groups, resource types, and resources. The scsnapshot tool formats the configuration data as a shell script that can be used for the following purposes:

- To replicate configuration data on a cluster that has no configured resource groups, resource types, and resources
- To upgrade configuration data on a cluster that has configured resource groups, resource types, and resources

The scsnapshot tool retrieves configuration data only from the Cluster Configuration Repository (CCR). Other configuration data is ignored. The scsnapshot tool does not take into account the dynamic state of different resource groups, resource types, and resources.

USAGE

This section describes how you can use the scsnapshot tool.

Retrieving Configuration Data for Resource Groups, Resource Types, and Resources

scsnapshot [-s *scriptfile*] [-o *imagefile*]

Used without the -o option, the scsnapshot tool generates a script that creates configuration data for clusters that do not already have configured resource groups, resource types, and resources. See Example 1.

Used with the -o option, the scsnapshot tool produces an image file that represents the configuration data. The image file can be used in further invocations of the scsnapshot tool to upgrade configuration data on a cluster. See Example 2.

To use the scsnapshot tool to retrieve configuration data, you need solaris.cluster.resource.read role-based access control (RBAC) authorization. For more information, see the rbac(5) man page.

To track differences between versions of configuration data, store the image files in a source control system such as SCCS.

Upgrading Configuration Data for Resource Groups, Resource Types, and Resources

scsnapshot [-s *scriptfile*] *oldimage newimage*

The scsnapshot tool generates a shell script that can be used to upgrade the configuration data that is contained in the *oldimage* file with the configuration data that is contained in the *newimage* file.

To use the scsnapshot tool to upgrade configuration data, you do not need specific RBAC authorization.

OPTIONS

The following options are supported by the scsnapshot tool. If you use an incorrect command option, the correct way to use the command option is displayed.

`-s` *scriptfile*
Stores the generated script in a file called *scriptfile*.

If this option is not specified, the generated script is written to the standard output.

If a file called *scriptfile* already exists, it is renamed as *scriptfile*`.old`, and a new file called *scriptfile* is created. If a file called *scriptfile*`.old` already exists, it is overwritten.

`-o` *imagefile*
Stores the generated image file in a file called *imagefile*.

If this option is not specified, an image file is not generated.

If a file called *imagefile* already exists, it is renamed as *imagefile*`.old`, and a new file called *imagefile* is created. If a file called *imagefile*`.old` already exists, it is overwritten.

oldimage
Specifies an image file that contains the old configuration data.

newimage
Specifies an image file that contains the new configuration data.

EXTENDED DESCRIPTION

The output of the `scsnapshot` tool is an executable Bourne-shell based script. Before you run the script, you might need to manually change some properties to reflect the features of your host.

The script compares the following characteristics of the local cluster to the cluster where the script was generated:

- Machine architecture
- Version of the Solaris Operating System
- Version of the Sun Cluster software

If the characteristics are not the same, the script writes an error and ends. A message asks whether you want to rerun the script by using the `-f` option. The `-f` option forces the script to run, despite any difference in characteristics.

The script generated by the `scsnapshot` tool verifies that the Sun Cluster resource type exists on the local cluster. If the resource type does not exist on the local cluster, the script writes an error and ends. A message asks whether you want to install the missing resource type before you run the script again.

To run a script that is generated by the `scsnapshot` tool, you need `solaris.cluster.resource.modify` RBAC authorization. For more information, see the `rbac`(5) man page.

EXAMPLES

EXAMPLE 1 To Generate a Shell Script That Retrieves Configuration Data for Resources Groups, Resource Types, and Resources

The script that is generated in this example is called `scriptfile.sh`.

```
example% scsnapshot -s scriptfile.sh
```

EXAMPLE 2 To Generate a Shell Script That Retrieves Configuration Data and Stores an Image File

The script that is generated in this example is called `scriptfile.sh`. The configuration data is stored an image file called `imagefile`.

```
example% scsnapshot -s scriptfile.sh -o imagefile
```

EXAMPLE 3 To Generate a Shell Script That Upgrades Configuration Data on One Cluster With Configuration Data From Another Cluster

This example creates a script that upgrades the configuration data on `cluster1` to match the configuration data on `cluster2`. The configuration data for `cluster1` is in a file called `imagefile1`, and the configuration data for `cluster2` is in a file called `imagefile2`. The name of a shell script is not specified, so the generated script is written to the standard output.

```
example% scsnapshot imagefile1 imagefile2
```

EXIT CODES

The following exit values are returned:

`0`	The command completed successfully.
`nonzero`	An error occurred. Error messages are displayed on the standard output.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu
Interface Stability	Evolving

SEE ALSO

`scrgadm`(1M), `r_properties`(5), `rbac`(5), `rg_properties`(5), `rt_properties`(5)

NAME

scstat – monitor the status of Sun Cluster

SYNOPSIS

scstat [-DWginpv [v]q] [-h *node*]

DESCRIPTION

The scstat command displays the current state of Sun Cluster and its components. Only one instance of the scstat command needs to run on any machine in the Sun Cluster configuration.

When run without any options, scstat displays the status for all components of the cluster. This display includes the following information:

- A list of cluster members
- The status of each cluster member
- The status of resource groups and resources
- The status of every path on the cluster interconnect
- The status of every disk device group
- The status of every quorum device
- The status of every IP Network Multipathing group and public network adapter

You need solaris.cluster.device.read, solaris.cluster.transport.read, solaris.cluster.resource.read, solaris.cluster.node.read, solaris.cluster.quorum.read, and solaris.cluster.system.read RBAC authorization to use this command without options. See rbac(5).

Resources and Resource Groups

The resource state, resource group state, and resource status are all maintained on a per-node basis. For example, a given resource has a distinct state on each cluster node and a distinct status on each cluster node.

The resource state is set by the Resource Group Manager (RGM) on each node, based only on which methods have been invoked on the resource. For example, after the STOP method has run successfully on a resource on a given node, the resource's state will be OFFLINE on that node. If the STOP method exits nonzero or times out, then the state of the resource is Stop_failed.

Possible resource states include: Online, Offline, Start_failed, Stop_failed, Monitor_failed, Online_not_monitored, Starting, and Stopping.

Possible resource group states are: Unmanaged, Online, Offline, Pending_online, Pending_offline, Error_stop_failed, Online_faulted, and Pending_online_blocked.

In addition to resource state, the RGM also maintains a resource status that can be set by the resource itself by using the API. The field Status Message actually consists of two components: status keyword and status message. Status message is optionally set by the resource and is an arbitrary text string that is printed after the status keyword.

Descriptions of possible values for a resource's status are as follows:

DEGRADED	The resource is online, but its performance or availability might be compromised in some way.

FAULTED	The resource has encountered an error that prevents it from functioning.
OFFLINE	The resource is offline.
ONLINE	The resource is online and providing service.
UNKNOWN	The current status is unknown or is in transition.

Device Groups

Device group status reflects the availability of the devices in that group.

The following are possible values for device group status and their descriptions:

DEGRADED	The device group is online, but not all of its potential primaries (secondaries) are up. For two-node connectivity, this status basically indicates that a stand-by primary does not exist, which means a failure of the primary node will result in a loss of access to the devices in the group.
OFFLINE	The device group is offline. There is no primary node. The device group must be brought online before any of its devices can be used.
ONLINE	The device group is online. There is a primary node, and devices within the group are ready for I/O.
WAIT	The device group is between one status and another. This status might occur, for example, when a device group is going from offline to online.

IP Network Multipathing Groups

IP Network Multipathing group status reflects the availability of the backup group and the adapters in the group.

The following are possible values for IP Network Multipathing group status and their descriptions:

OFFLINE	The backup group failed. All adapters in the group are offline.
ONLINE	The backup group is functional. At least one adapter in the group is online.
UNKNOWN	Any other state than those listed before. This could result when an adapter is detached or marked as down by Solaris commands such as `if_mpadm`(1M) or `ifconfig`(1M).

The following are possible values for IP Network Multipathing adapter status and their descriptions:

OFFLINE	The adapter failed or the backup group is offline.
ONLINE	The adapter is functional.

`STANDBY`	The adapter is on standby.
`UNKNOWN`	Any other state than those listed before. This could result when an adapter is detached or marked as down by Solaris commands such as `if_mpadm` or `ifconfig`.

OPTIONS

You can specify command options to request the status for specific components.

If more than one option is specified, the `scstat` command prints the status in the specified order.

The following options are supported:

`-D`	Shows status for all disk device groups. You need `solaris.cluster.device.read` RBAC authorization to use this command option. See `rbac`(5).
`-g`	Shows status for all resource groups. You need `solaris.cluster.resource.read` RBAC authorization to use this command option. See `rbac`(5).
`-h` *node*	Shows status for the specified node (*node*) and status of the disk device groups of which this *node* is the primary node. Also shows the status of the quorum devices to which this node holds reservations of the resource groups to which the *node* is a potential master, and holds reservations of the transport paths to which the *node* is attached. You need `solaris.cluster.device.read`, `solaris.cluster.transport.read`, `solaris.cluster.resource.read`, `solaris.cluster.node.read`, `solaris.cluster.quorum.read`, and `solaris.cluster.system.read` RBAC authorization to use this command option. See `rbac`(5).
`-i`	Shows status for all IP Network Multipathing groups and public network adapters.
`-n`	Shows status for all nodes. You need `solaris.cluster.node.read` RBAC authorization to use this command option. See `rbac`(5).
`-p`	Shows status for all components in the cluster. Use with `-v` to display more verbose output.

You need `solaris.cluster.device.read`, `solaris.cluster.transport.read`, `solaris.cluster.resource.read`, `solaris.cluster.node.read`, `solaris.cluster.quorum.read`, and `solaris.cluster.system.read` RBAC authorization to use `-p` with `-v`. See `rbac`(5).

`-q` Shows status for all device quorums and node quorums.

You need `solaris.cluster.quorum.read` RBAC authorization to use this command option. See `rbac`(5).

`-v[v]` Shows verbose output.

`-W` Shows status for cluster transport path.

You need `solaris.cluster.transport.read` RBAC authorization to use this command option. See `rbac`(5).

EXAMPLES

EXAMPLE 1 Using the `scstat` Command

The following command displays the status of all resource groups followed by the status of all components related to the specified host:

```
% scstat -g -h host
```

The output that is displayed appears in the order in which the options are specified.

These results are the same results you would see by typing the two commands:

```
% scstat -g
```

and

```
% scstat -h host
```

EXIT STATUS

The following exit values are returned:

`0` The command completed successfully.

nonzero An error has occurred.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWcsu`
Interface Stability	Evolving

SEE ALSO

`scha_resource_setstatus`(1HA), `scha_resource_setstatus`(3HA), `attributes`(5)

NOTES An online quorum device means that the device was available for contributing to the formation of quorum when quorum was last established. From the context of the quorum algorithm, the device is online because it actively contributed to the formation of quorum. However, an online quorum device might not necessarily continue to be in a healthy enough state to contribute to the formation of quorum when quorum is re-established. The current version of Sun Cluster does not include a disk monitoring facility or regular probes to the quorum devices.

NAME

scswitch – perform ownership and state change of resource groups and disk device groups in Sun Cluster configurations

SYNOPSIS

scswitch -c -h *node[,...]* -j *resource[,...]* -f *flag-name*

scswitch {-e | -n} [-M] -j *resource[,...]*

scswitch -F {-g *resource-grp[,...]* | -D *device-group[,...]*}

scswitch -m -D *device-group[,...]*

scswitch -Q [-g *resource-grp[,...]*]

scswitch -R -h *node[,...]* -g *resource-grp[,...]*

scswitch -S -h *from-node* [-K *continue_evac*]

scswitch {-u | -o} -g *resource-grp[,...]*

scswitch -z -g *resource-grp[,...]* -h *node[,...]*

scswitch -z -g *resource-grp[,...]*

scswitch -z

scswitch -z -D *device-group[,...]* -h *node*

scswitch -Z [-g *resource-grp[,...]*]

DESCRIPTION

The scswitch command moves resource groups or disk device groups to new primary nodes. It also provides options for evacuating all resource groups and disk device groups from a node by moving ownership elsewhere, bringing resource groups or disk device groups offline and online, enabling or disabling resources, switching resource groups to or from an Unmanaged state, or clearing error flags on resource groups.

You can run the scswitch command from any node in a Sun Cluster configuration. If a device group is offline, you can use scswitch to bring the device group online onto any host in the node list. However, once the device group is online, a switchover to a spare node is not permitted. Only one invocation of scswitch at a time is permitted.

Do not attempt to kill an scswitch operation that is already underway.

There are ten forms of the scswitch command, each specified by a different option. See SYNOPSIS and OPTIONS.

change error flag (-c)
Clears the specified error *flag-name* on one or more resources on the specified *node*s.

enable or disable (-e or -n)
Enables or disables the specified *resources*.

take offline (-F)
Takes the specified *resource-grp*s or *device-grp*s offline on all nodes.

set maintenance mode (-m)
Takes the specified disk *device-grp*s offline from the cluster for maintenance. The resulting state survives reboots. If a disk device group is currently being accessed,

this action fails and the specified disk device groups are not taken offline from the cluster. Disk device groups are brought back online by using the `-z` option. Only explicit calls to `scswitch` can bring a disk device group out of maintenance mode.

quiesce (`-Q`)
Brings the specified *resource-grp*s to a quiescent state. This option stops these *resource-grp*s from continuously bouncing around from one node to another in the event of the failure of a `START` or `STOP` method.

restart (`-R`)
Takes the specified *resource-grp*s offline and then back online on the specified primary *node*s of the resource groups. The specified *node*s must be the current primaries of the resource groups.

evacuate or switch all (`-S`)
Attempts to switch over all resource groups and disk device groups from the specified *from-node* to a new set of primaries. The system attempts to select new primaries based on configured preferences for each group. All evacuated groups are not necessarily remastered by the same primary. If one or more resource groups or disk device groups cannot be evacuated from the specified *from-node*, the command fails, issues an error message, and exits with a nonzero exit code.

unmanage or manage (`-u` or `-o`)
Takes the specified *resource-grp*s to (`-u`) the unmanaged state or takes the specified unmanaged *resource-grp*s out of (`-o`) the unmanaged state.

The `-o` option brings the specified *resource-grp*s under Resource Group Manager (RGM) management so that the RGM attempts to bring the resource groups online.

set primaries (`-z`)
Causes the orderly transfer of one or more *resource-grp*s or disk *device-grp*s from one primary node in a Sun Cluster configuration to another node in the configuration (or to multiple nodes for resource groups that are configured with multiple primaries). This option takes resource groups offline and brings disk device groups back online after being in maintenance mode. This option also brings all or selected resource groups online on their most-preferred node or nodes. This option does not, however, enable any resources, enable monitoring on any resources, or take any resource groups out of the unmanaged state, as the `-Z` option does.

bring online (`-Z`)
Enables all resources in the specified *resource-grp*s, enables monitoring on all resources, manages groups, and brings the groups online on the default list of primaries.

OPTIONS

The ten forms of the `scswitch` command are specified by the following options:

`-c`
Clears the `-f` *flag-name* on the specified set of *resource*s on the specified *node*s. For the current release of Sun Cluster software, the `-c` option is only implemented for the `Stop_failed` error flag. Clearing the `Stop_failed` error flag places the resource into the offline state on the specified *node*s.

If the `Stop` method fails on a resource and the `Failover_mode` property of the resource is set to `Hard`, the RGM halts or reboots the node to force the resource (and all other resources mastered by that node) offline.

If the `Stop` method fails on a resource and the `Failover_mode` property is set to a value other than `Hard`, the individual resource goes into the `Stop_failed` state and the resource group is placed into the `Error_stop_failed` state. A resource group in the `Error_stop_failed` state on any node cannot be brought online on any node, nor can it be edited (you cannot add or delete resources or change resource group properties or resource properties). You must clear the `Stop_failed` state by performing the procedure documented in the *Sun Cluster Data Services Installation Guide for Solaris OS*.

Caution – Make sure that both the resource and its monitor are stopped on the specified *node* before you clear the `Stop_failed` flag. Clearing the `Stop_failed` error flag without fully killing the resource and its monitor can lead to more than one instance of the resource executing on the cluster simultaneously. If you are using shared storage, this situation can cause data corruption. If necessary, as a last resort, execute a `kill`(1) command on the associated processes.

`-e` or `-n` Enables (`-e`) or disables (`-n`) the specified *resource*s.

You cannot disable a resource without also disabling all resources that depend on that resource. Conversely, you cannot enable a resource unless all of the resources on which that resource depends are also enabled. Once you have enabled a resource, it goes online or offline depending on whether its resource group is online or offline. A disabled resource is immediately brought offline from all of its current masters and remains offline regardless of the state of its resource group.

`-F` Takes the specified *resource-grp*s (`-g`) or *device-group*s (`-D`) offline on all nodes.

When the `-F` option takes a disk device group offline, the associated VxVM disk group or Solstice DiskSuite diskset is unported or released by the primary node. Before a disk device group can be taken offline, all access to its devices must be stopped and all dependent file systems must be unmounted. You must start an offline disk device group by issuing an explicit `scswitch` call, by accessing a device within the group, or by mounting a file system that depends on the group.

`-m` Specifies the “set maintenance mode” form of the `scswitch` command.

The `-m` option takes the specified *device-group*s offline from the cluster for maintenance. Before a disk device group can be placed in maintenance mode, all access to its devices must be stopped and all dependent file systems must be unmounted. Disk device groups are brought back online by using the `-z` option.

`-Q` Brings the specified *resource-grp*s, which might be reconfigured, to a quiescent state. This form of the `scswitch` command does not exit until the *resource-grp*s have reached a quiescent state in which they are no longer stopping or starting on any node.

If a `Monitor_stop`, `Stop`, `Postnet_stop`, `Start`, or `Prenet_start` method fails, on any resource in a group while the `scswitch -Q` command is executing, the resource behaves as if its `Failover_mode` property was set to `None`, regardless of its actual setting. Upon failure of one of these methods, the resource moves to an error state (either `Start_failed` or `Stop_failed`) rather than initiating a failover or a rebooting of the node.

When the `scswitch -Q` command exits, the specified *resource-grp*s might be online or offline. You can determine their current state by executing the `scstat(1M)` command.

If a node dies during execution of the `scswitch -Q` command, execution might be interrupted, and, as a result, the resource groups are left in a non-quiescent state. If execution is interrupted, `scswitch -Q` returns a nonzero exit code and writes an error message to the standard error. In this case, you can re-issue the `scswitch -Q` command.

`-R` Specifies the “restart” form of the command. The `-R` option moves the specified *resource-grp*s offline and then back online on the specified primary *node*s. The resource groups must already be mastered by all of the specified nodes.

`-S` Specifies the “evacuate” or “switch all” form of the `scswitch` command.

The `-S` option switches all resource groups and disk device groups off the specified *node*. If not all groups owned by the given node can be successfully evacuated to a new set of primaries, the command exits with an error. If the primary ownership of a group cannot be changed to one of the other nodes, primary ownership for that group is retained by the original node.

`-u` or `-o` Specifies the “change resource group state” form of the `scswitch` command.

The `-u` option takes the specified managed *resource-grp*s to the unmanaged state. As a precondition of the `-u` option, all resources that belong to the indicated resource groups must first be disabled.

The `-o` option takes the specified unmanaged *resource-grp*s to the managed state. Once a resource group is in the managed state, the RGM attempts to bring the resource group online.

`-z` Specifies a change in mastery of a specified *resource-grp* or a disk *device-grp*.

When used with the `-g` and `-h` options, the `-z` option brings the specified *resource-grp*s online on the *node*s specified by the `-h` option and takes them offline on all other cluster nodes. If the node list specified with the `-h` option is the empty set, the `-z` option takes the resource groups specified by the `-g` option offline from all of their current masters. If one of the listed *resource-grp*s is not capable of being mastered by *node*, an error is reported and no *resource-grp*s are switched over. All nodes specified by the `-h` option must be current members of the cluster and must be potential primaries of all of the resource groups specified by the `-g` option. The number of nodes specified by the `-h` option must not exceed the setting of the `Maximum_primaries` property of any of the resource groups specified by the `-g` option.

When used with only the `-g` option, the `-z` option brings the specified *resource-grp*s, which must already be managed, online on their most-preferred node or nodes. This form of `scswitch` does not bring a resource group online in violation of its strong `RG_affinities`, and writes a warning message if the affinities of a resource group cannot be satisfied on any node.

If you configure the `RG_affinities` properties of one or more resource groups, and you issue the **`scswitch -z -g`** command (with or without the `-h` option), additional resource groups other than those that are specified after the `-g` option might be switched as well. `RG_affinities` is described in `rg_properties`(5).

When used alone (**`scswitch -z`**), the `-z` switches all managed resource groups online on their most-preferred node or nodes.

When used with only `-g` or when used alone, the `-z` option only switches resources and groups online, unlike the `-Z` option. Resource groups that are unmanaged remain unmanaged, and resources that are disabled or that have monitoring disabled are left in the disabled state.

When used with the `-D` option, the `-z` option switches one or more specified *device-group*s to the specified *node*. Only one primary node name can be specified for a disk device group's switchover. When multiple *device-group*s are specified, the `-D` option switches the *device-group*s in the order specified. If the `-z -D` operation encounters an error, the operation stops and no further switches are performed.

`-Z` Enables all resources of the specified *resource-grp*s and their monitors, moves the *resource-grp* into the managed state, and brings the *resource-grp* online on all the default primaries. When the `-g` option is not specified, the `scswitch` command attempts to bring all resource groups online.

You can combine the following options with the previous ten options as follows:

`-D` Specifies the name of one or more *device-group*s.

This option is only legal with the `-F`, `-m`, and `-z` options.

You need `solaris.cluster.device.admin` RBAC authorization to use this command option with `-F`, `-m`, and `-z` (in conjunction with `-h`). See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

`-f` Specifies the error *flag-name*.

This option is only legal with the `-c` option.

The only error flag currently supported is `Stop_failed`.

You need `solaris.cluster.resource.admin` RBAC authorization to use this command option with `-c`. See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

`-g` Specifies the name of one or more *resource-grp*s.

This option is only legal with the `-F`, `-o`, `-Q`, `-R`, `-u`, `-z`, and `-Z` options.

You need `solaris.cluster.resource.admin` RBAC authorization to use this command option with `-F`, `-o`, `-R` (in conjunction with `-h`), `-u`, `-z` (in conjunction with `-h`), or `-Z`. See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

`-h` Specifies the names of one or more cluster *node*s.

This option is only legal with the `-c`, `-R`, `-S`, and `-z` options.

When used with the `-c`, `-R`, or `-z` option, the `-h` option specifies the target server (or list of servers in the case of resource groups configured with multiple primaries).

When used with the `-S` option, the `-h` option specifies the original server. A comma-delimited list of *node*s can be specified after the `-h` option for *resource-grp*s or *device-group*s that are configured with multiple primaries. In this case, if any of the listed primaries cannot master a particular *resource-grp* or *device-group*, *resource-grp* or disk *device-group* is not switched over.

You need `solaris.cluster.resource.admin` RBAC authorization to use this command option with `-c`, `-R` (in conjunction with `-g`), `-S`, and `-z` (in conjunction with `-g`). In addition, you need `solaris.cluster.device.admin` RBAC authorization to use this command option with `-z` (in conjunction with `-D`). See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

`-j` Specifies the names of one or more *resource*s.

This option is legal only with the `-c`, `-e`, and `-n` options.

You need `solaris.cluster.resource.admin` RBAC authorization to use this command option with `-c`, `-e`, or `-n`. See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

`-K` Specifies the number of seconds to keep resource groups from switching back onto a node after that node has been successfully evacuated.

Resource groups cannot fail over or automatically switch over onto the node while that node is being evacuated, and, after evacuation is completed, for the number of seconds that you specify with this option. You can, however, initiate a switchover onto the evacuated node with the **`scswitch -z -g -h`** command before *continue_evac* seconds have passed. Only automatic switchovers are prevented.

This option is legal only with the `-S` option. You must specify an integer value between 0 and 65535. If you do not specify a value, 60 seconds is used by default.

You need `solaris.cluster.resource.admin` RBAC authorization to use this command option. See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

`-M` Enables (`-e`) or disables (`-n`) monitoring for the specified resources. When you disable a resource, you need not disable monitoring on it because both the resource and its monitor are kept offline.

This option is legal only with the `-e` and `-n` options.

You need `solaris.cluster.resource.admin` RBAC authorization to use this command option with `-e` and `-n`. See `rbac`(5).

You must also be able to assume a role to which the Sun Cluster Commands rights profile has been assigned to use this command. Authorized users can issue privileged Sun Cluster commands on the command line from the `pfsh`(1), `pfcsh`(1), or `pfksh`(1) profile shell. A profile shell is a special kind of shell that enables you to access privileged Sun Cluster commands that are assigned to the Sun Cluster Commands rights profile. A profile shell is launched when you run `su`(1M) to assume a role. You can also use `pfexec`(1) to issue privileged Sun Cluster commands.

EXAMPLES

EXAMPLE 1 Switching Over a Resource Group

The following command switches over *resource-grp-2* to be mastered by *node1*:

```
node1# scswitch -z -h node1 -g resource-grp-2
```

EXAMPLE 2 Switching Over a Managed Resource Group Without Enabling Monitoring or Resources

The following command brings *resource-grp-2* online if *resource-grp-2* is already managed, but does not enable any resources or enable monitoring on any resources that are currently disabled.

```
node1# scswitch -z -g resource-grp-2
```

EXAMPLE 3 Switching Over a Resource Group Configured to Have Multiple Primaries

The following command switches over *resource-grp-3*, a resource group configured to have multiple primaries, to be mastered by *node1*,*node2*,*node3*:

```
node1# scswitch -z -h node1,node2,node3 -g resource-grp-3
```

EXAMPLE 4 Moving All Resource Groups and Disk Device Groups Off of a Node

The following command switches over all resource groups and disk device groups from *node1* to a new set of primaries:

```
node1# scswitch -S -h node1
```

EXAMPLE 5 Moving All Resource Groups and Disk Device Groups Persistently Off of a Node

The following command switches over all resource groups and disk device groups from *node1* to a new set of primaries. The following command also shows how to prevent resource groups from automatically switching back onto that node after that node has been successfully evacuated. For example, this situation might occur if one of the resource groups failed to start on its new master. You prevent this situation from occurring by setting the `-K` option *continue_evac* to an integer number of seconds, in

EXAMPLE 5 Moving All Resource Groups and Disk Device Groups Persistently Off of a Node *(Continued)*

this example, two minutes. That is, by setting `-K` to 120, you prevent resource groups from switching back onto the evacuated node for two minutes. This situation arises when resource groups attempt to switch back automatically when strong negative affinities have been configured (with `RG_affinities`).

```
node1# scswitch -S -h node1 -K 120
```

EXAMPLE 6 Restarting Some Resource Groups

The following command restarts some resource groups on the specified nodes:

```
node1# scswitch -R -h node1,node2 -g resource-grp-1,resource-grp-2
```

EXAMPLE 7 Disabling Some Resources

```
node1# scswitch -n -j resource-1,resource-2
```

EXAMPLE 8 Enabling a Resource

```
node1# scswitch -e -j resource-1
```

EXAMPLE 9 Taking Resource Groups to the Unmanaged State

```
node1# scswitch -u -g resource-grp-1,resource-grp-2
```

EXAMPLE 10 Taking Resource Groups Out of the Unmanaged State

```
node1# scswitch -o -g resource-grp-1,resource-grp-2
```

EXAMPLE 11 Switching Over a Device Group

The following command switches over *device-group-1* to be mastered by *node2*:

```
node1# scswitch -z -h node2 -D device-group-1
```

EXAMPLE 12 Putting a Device Group Into Maintenance Mode

The following command puts *device-group-1* into maintenance mode:

```
node1# scswitch -m -D device-group-1
```

EXAMPLE 13 Quiescing Resource Groups

The following command brings resource groups RG1 and RG2 to a quiescent state:

```
node1# scswitch -Q -g RG1,RG2
```

EXIT STATUS

This command blocks until requested actions are completely finished or an error occurs.

The following exit values are returned:

`0`	The command completed successfully.
nonzero	An error has occurred. `scswitch` writes an error message to standard error.

If `scswitch` exits nonzero with the error message `cluster is reconfiguring`, the requested operation might have completed successfully, despite the error status. If you doubt the result, you can execute `scswitch` again with the same arguments after the reconfiguration is complete.

If `scswitch` exits nonzero with the error message `Resource group failed to start on chosen node and may fail over to other node(s)`, the resource group will continue to reconfigure for some time after the `scswitch` command exits. Additional `scswitch` or `scrgadm(1M)` operations on that resource group will fail until the resource group has reached a terminal state such as `Online`, `Online_faulted`, or `Offline` on all nodes.

If you invoke the `scswitch` command on multiple resource groups and multiple errors occur, the exit value only reflects one of the errors. To avoid this possibility, invoke `scswitch` on just one resource group at a time.

Some operations are not permitted on a resource group (and its resources) whose `RG_system` property is `True`. See `rg_properties(5)` for more information.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWcsu`
Interface Stability	Evolving

SEE ALSO

`kill`(1), `scrgadm(1M)`, `scstat(1M)`, `attributes`(5), `rg_properties(5)`

Sun Cluster Data Services Installation Guide for Solaris OS

NOTES

If you take a resource group offline by using the `-z` or `-F` options with the `-g` option, the `Offline` state of the resource group will not survive node reboots. In other words, if a node dies or joins the cluster, the resource group might come online on some node, even if you previously switched the resource group offline. Even if all of the resources are disabled, the resource group will come online. To prevent the resource group from coming online, you must either put the resource group in the `Unmanaged` state or set the `Desired_primaries` property of the group to zero.

NAME

scversions – Sun Cluster version management

SYNOPSIS

```
scversions [-c]
```

DESCRIPTION

The `scversions` command commits the cluster to a new level of functionality after a rolling–upgrade to new Sun Cluster software. With no arguments, the `scversions` command prints a message indicating whether a commitment is needed.

OPERANDS

The following operands are supported:

`-c` Commit the set of nodes that are currently active members of the cluster to the highest possible level of functionality.

When you upgrade a node (either through upgrade to a new release of the product or by application of a patch) and boot it back into the cluster, some of the internal protocols on that node might have to run at lower versions in order to cooperate correctly with other nodes in the cluster. When the cluster is in this state, some administrative actions might be disabled and some new functionality introduced in the upgrade might be unavailable.

When you run this command once from any node after all nodes is upgraded, the cluster switches to the highest versions of internal protocols possible. Assuming all nodes have the same Sun Cluster software installed at that time, all new functionality becomes available and any administrative restrictions are removed.

If a node that has not been upgraded is an active member of the cluster at the time you run the `-c` option to `scversions`, the command has no effect because the cluster is already running at the highest possible level of functionality.

If a node has not been upgraded and is not an active member of the cluster when you run the `-c` option to `scversions` (for example, if that node is down for maintenance), the internal protocols of the cluster are upgraded to the highest possible versions. You might have to upgrade the node that was not an active member of the cluster to enable it to rejoin the cluster.

EXIT STATUS

0 Success

non-zero Failure

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu
Interface Stability	Evolving

SEE ALSO

`scinstall`(1M)

Sun Cluster 3.1 5/03 Software Installation Guide

NAME

scvxinstall – install VERITAS Volume Manager (VxVM) on a cluster node

SYNOPSIS

scvxinstall [-d *cdrom-image*] [-L *license*...]

scvxinstall {-i | -e} [-d *cdrom-image*] [-L *license*...]

scvxinstall -s

scvxinstall -H

DESCRIPTION

The scvxinstall utility provides automatic VxVM installation and optional root-disk encapsulation for Sun Cluster nodes.

The first form of the scvxinstall utility in the SYNOPSIS section of this man page runs in interactive mode. All other forms of the utility run in non-interactive mode.

- In interactive mode, scvxinstall prompts the user for the mode of operation ("install only" or "install and encapsulate") and for any needed CD-ROM and licensing information.
- In non-interactive mode, scvxinstall does not prompt the user for information. If any needed information is not supplied on the utility line, scvxinstall terminates with an error return code.

The cluster must meet the following requirements before you run the scvxinstall utility:

- All nodes in the cluster configuration must be current cluster members.
- Each root disk that you will encapsulate must have at least two free (unassigned) partitions.
- All nodes must be added to the node authentication list.
- As of VxVM 4.1, you must install VxVM software and licenses before you run the scvxinstall utility.

The "install-only" mode of the scvxinstall utility performs the following tasks:

1. Verifies that the node you are installing is booted in cluster mode and is running as root, and verify that all other cluster nodes are running in cluster mode.
2. For VxVM 4.0 or earlier, adds the VxVM software, licensing, and man-page packages, but not the GUI packages.
3. Negotiates a cluster-wide value for the vxio major number by modifying the /etc/name_to_major file. This ensures that the vxio number is the same on all cluster nodes.
4. For VxVM 4.0 or earlier, installs the VxVM license key.
5. Instructs the user to reboot the node to resume operation with the new vxio major numbers in effect.

The "install-and-encapsulate" mode of the scvxinstall utility performs the same tasks as the "install-only" mode **except** Step 5, then performs the following additional tasks:

1. Runs several VxVM commands to prepare for root-disk encapsulation.

2. Modifies the global-devices entry in the `/etc/vfstab` file specified for the `/global/.devices/node@`*n* file system, where *n* is the node ID number. The `scvxinstall` utility replaces the existing device path `/dev/did/{r}dsk` with `/dev/{r}dsk`. This change ensures that VxVM recognizes that the global-devices file system resides on the root disk.
3. Twice reboots each node that is running `scvxinstall`, once to allow VxVM to complete the encapsulation process and once more to resume normal operation. The `scvxinstall` utility includes a synchronization mechanism to ensure that it reboots only one node at a time, to prevent loss of quorum.
4. Unmounts the global-devices file system. The file system is automatically remounted after the encapsulation process is complete.
5. Recreates the special files for the root-disk volumes with a unique minor number on each node.

OPTIONS

The following options are supported:

`-d` *cdrom-image*
Valid only for VxVM 4.0 or earlier. Specifies the path to the VxVM packages.

`-e`
Specifies the "install and encapsulate" mode of the `scvxinstall` utility. This option installs VxVM, if installing VxVM 4.0 or earlier, encapsulates the root disk, and performs postinstallation tasks. If the `scvxinstall` utility was previously run on the node in "install only" mode, `scvxinstall` confirms that "install only" mode tasks are completed before it performs the root-disk encapsulation tasks.

`-H`
Specifies the "help" mode of the `scvxinstall` utility. This option displays a brief help message about the `scvxinstall` utility.

`-i`
Specifies the “install only” mode of the `scvxinstall` utility. This option installs VxVM, if installing VxVM 4.0 or earlier, and performs postinstallation tasks, but does not encapsulate the root disk.

`-L` *license*
Valid only for VxVM 4.0 or earlier. Specifies a license key for the VxVM software. You can specify the `-L` *license* option multiple times to supply multiple license keys to the `scvxinstall` utility. If you have no additional license keys to install, you can specify the word `none` for the *license* argument to the `-L` option.

`-s`
Specifies the “show status” mode of the `scvxinstall` utility. This option displays the status of running or completed `scvxinstall` processing on the node.

EXAMPLES

EXAMPLE 1 Running `scvxinstall` Interactively

The following command runs `scvxinstall` interactively.

```
example#  scvxinstall
```

EXAMPLE 2 Installing the VxVM Packages Without Encapsulating the Root Disk

The following command installs the VxVM 4.0 packages but does not encapsulate the root disk. This command also supplies the VxVM license key. This example assumes that the VxVM CD-ROM is in the CD-ROM drive.

```
example#  scvxinstall -i -L "9999 9999 9999 9999 9999 999"
```

EXAMPLE 3 Installing the VxVM Packages Without Encapsulating the Root Disk

The following command installs the VxVM 4.0 packages but does not encapsulate the root disk. The command supplies the path to the CD-ROM images of the VxVM packages, which are stored on a server.

```
example#  scvxinstall -i -d /net/myserver/VxVM/pkgs
```

EXAMPLE 4 Installing the VxVM Packages and Encapsulating the Root Disk

The following command installs the VxVM 4.0 packages and encapsulates the root disk. The command supplies the VxVM license key. This example assumes that the VxVM CD-ROM is in the CD-ROM drive.

```
example#  scvxinstall -e -L "9999 9999 9999 9999 9999 999"
```

EXAMPLE 5 Installing the VxVM Packages and Encapsulating the Root Disk

The following command installs the VxVM 4.0 packages and encapsulates the root disk. The command supplies the path to the CD-ROM images and supplies the VxVM license key.

```
example# scvxinstall -e -d /net/myserver/VxVM/pkgs -L "9999 9999 9999 9999 9999 999"
```

EXAMPLE 6 Encapsulating the Root Disk After Installing VxVM 4.1 Software

The following command verifies that VxVM 4.1 software and licenses are installed and configured, encapsulates the root disk, and negotiates a cluster-wide value for the `vxio` major number.

```
example#  scvxinstall -e
```

EXAMPLE 7 Performing Postinstallation Tasks After Installing VxVM 4.1 Software

The following command verifies that VxVM 4.1 software and licenses are installed and configured, then negotiates a cluster-wide value for the `vxio` major number.

```
example#  scvxinstall -i
```

EXIT STATUS

The following exit values are returned:

`0`	Successful completion.
non-zero	An error has occurred.

FILES

`/etc/rc2.d/S74scvxinstall.sh`
(Solaris 8 and Solaris 9) An `rc` script used to complete processing following a root-disk-encapsulation reboot

`/usr/cluster/lib/svc/method/scvxinstall`
(Solaris 10) An `rc` script used to complete processing following a root-disk-encapsulation reboot

`/var/cluster/logs/install/scvxinstall.log.`*pid*
Log file created by `scvxinstall`

`/var/cluster/scvxinstall/*`
Location of temporary files used by `scvxinstall`

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcsu, SUNWscr
Interface Stability	Evolving

SEE ALSO

`scconf`(1M), `scinstall`(1M), `scsetup`(1M)

Sun Cluster Software Installation Guide for Solaris OS

SC31 3ha

NAME

scds_close – free DSDL environment resources

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_close(scds_handle_t *handle);
```

DESCRIPTION

The `scds_close()` function reclaims resources that were allocated during data service method initialization by using `scds_initialize(3HA)`. Call this function once, prior to termination of the program.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize()`.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
include file

`/usr/cluster/lib/libdsdev.so`
library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_initialize(3HA)`, `attributes`(5)

NAME

scds_error_string – translate an error code to an error string

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

const char *scds_error_string(scha_err_t error_code);
```

DESCRIPTION

The `scds_error_string()` function translates an error code from a DSDL function into a short string describing the error. Invalid error codes return `NULL`.

The pointer returned by this function is to memory belonging to the DSDL. Do not modify this memory.

PARAMETERS

The following parameters are supported:

error_code Error code returned by a DSDL function.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
include file

`/usr/cluster/lib/libdsdev.so`
library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scha_calls`(3HA), `attributes`(5)

NAME

scds_failover_rg – failover a resource group

SYNOPSIS

```
cc [flags
...] -I/usr/cluster/include file -L
/usr/cluster/lib -ldsdev

#include <rgm/libdsdev.h>

scha_err_t scds_failover_rg(scds_handle_t handle);
```

DESCRIPTION

The `scds_failover_rg()` function performs a `scha_control(3HA)` `SCHA_GIVEOVER` operation on the resource group containing the resource passed to the calling program.

When this function succeeds, it does not return. Therefore, treat this function as the last piece of code to be executed in the calling program.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize(3HA)`.

RETURN VALUES

The following return values are supported:

`SCHA_ERR_NOERR` Indicates the function succeeded.

Other values Indicate the function failed. See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
include file

`/usr/cluster/lib/libdsdev.so`
library

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scha_calls(3HA)`, `scha_control(3HA)`, `attributes(5)`

NAME

scds_fm_action – take action after probe completion

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_fm_action(scds_handle_t handle, int probe_status, long
     elapsed_milliseconds);
```

DESCRIPTION

The `scds_fm_action()` function uses the `probe_status` of the data service in conjunction with the past history of failures to take one of the following actions:

- Restart the application.
- Fail over the resource group.
- Do nothing.

Use the value of the input `probe_status` argument to indicate the severity of the failure. For example, you might consider a failure to connect to an application as a complete failure, but a failure to disconnect as a partial failure. In the latter case you would have to specify a value for `probe_status` between 0 and `SCDS_PROBE_COMPLETE_FAILURE`.

The DSDL defines `SCDS_PROBE_COMPLETE_FAILURE` as `100`. For partial probe success or failure, use a value between `0` and `SCDS_PROBE_COMPLETE_FAILURE`.

Successive calls to `scds_fm_action()` compute a failure history by summing the value of the `probe_status` input parameter over the time interval defined by the `Retry_interval` property of the resource. Any failure history older than `Retry_interval` is purged from memory and is not used towards making the restart or failover decision.

The `scds_fm_action()` function uses the following algorithm to choose which action to take:

Restart

If the accumulated history of failures reaches `SCDS_PROBE_COMPLETE_FAILURE`, `scds_fm_action()` restarts the resource by calling the `STOP` method of the resource followed by the `START` method. It ignores any `PRENET_START` or `POSTNET_STOP` methods defined for the resource type.

The status of the resource is set to `SCHA_RSSTATUS_DEGRADED` by making a `scha_resource_setstatus()` call, unless the resource is already set.

If the restart attempt fails because the `START` or `STOP` methods of the resource fail, a `scha_control()` is called with the `GIVEOVER` option to fail the resource group over to another node. If the `scha_control()` call succeeds, the resource group is failed over to another cluster node and the call to `scds_fm_action()` never returns.

Upon a successful restart, failure history is purged. Another restart is attempted if and only if the failure history again accumulates to SCDS_PROBE_COMPLETE_FAILURE.

Failover

If the number of restarts attempted by successive calls to scds_fm_action() reaches the Retry_count value defined for the resource, a failover is attempted by making a call to scha_control() with the GIVEOVER option.

The status of the resource is set to SCHA_RSSTATUS_FAULTED by making a scha_resource_setstatus() call, unless the resource is already set.

If the scha_control() call fails, the entire failure history maintained by scds_fm_action() is purged.

If the scha_control() call succeeds, the resource group is failed over to another cluster node and the call to scds_fm_action() never returns.

No Action

If the accumulated history of failures remains below SCDS_PROBE_COMPLETE_FAILURE, no action is taken. In addition, if the probe_status value is 0, which indicates a successful check of the service, no action is taken, irrespective of the failure history.

The status of the resource is set to SCHA_RSSTATUS_OK by making a scha_resource_setstatus() call, unless the resource is already set.

PARAMETERS

The following parameters are supported:

handle	The handle returned from scds_initialize(3HA).
probe_status	A number you specify between 0 and SCDS_PROBE_COMPLETE_FAILURE that indicates the status of the data service. A value of 0 implies that the recent data service check was successful. A value of SCDS_PROBE_COMPLETE_FAILURE means complete failure and implies that the service has completely failed. You can also supply a value in between 0 and SCDS_PROBE_COMPLETE_FAILURE that implies a partial failure of the service.
elapsed_milliseconds	The time, in milliseconds, to complete the data service check. This value is reserved for future use.

RETURN VALUES

The following exit values are returned:

0	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR` No action was taken, or a restart was successfully attempted.

`SCHA_ERR_FAIL` A failover attempt was made but it did not succeed.

`SCHA_ERR_NOMEM` System is out of memory.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
include file

`/usr/cluster/lib/libdsdev.so`
library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_fm_sleep`(3HA), `scds_initialize`(3HA), `scha_calls`(3HA), `scha_control`(3HA), `scha_fm_print_probes`(3HA), `scha_resource_setstatus`(3HA), `attributes`(5)

NAME

scds_fm_net_connect – establish a TCP connection to an application

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_fm_net_connect(scds_handle_t handle, scds_socket_t
    *socklist, int count, scds_netaddr_t addr, time_t timeout);
```

DESCRIPTION

The `scds_fm_net_connect()` function establishes one or more TCP connections (depending on the protocol value of `Port_list` for each address, as described below) to a process that is being monitored.

You can retrieve a list of network addresses for the resource by using `scds_get_netaddr_list(3HA)`. That call also fills the protocol value for each address in the list. If `tcp6` is specified as the protocol in `Port_list` for that address, the protocol value is set to `SCDS_IPPROTO_TCP6`. If `tcp` is specified as the protocol in `Port_list` for that address or if no protocol is specified in `Port_list`, the protocol value is set to `SCDS_IPPROTO_TCP`.

This function also resolves the `hostname` that is supplied in `addr` and connects to:

- The IPv4 address of the `hostname` at the specified port, if the protocol that is specified in `addr` is `SCDS_IPPROTO_TCP`.
- Both the IPv4 address (if there is one) and the IPv6 address (if there is one) of the `hostname` at the specified port, if the protocol specified in `addr` is `SCDS_IPPROTO_TCP6`. The status and the file descriptor, if applicable, are stored in the `scds_socket_t` array that is supplied to this function. The first member of this array is used for the IPv4 mapping and the second member of this array is used for IPv6. The status can be set to one of the following values:
  - `SCDS_FMSOCK_OK` — The operation succeeded and the associated socket file descriptor is valid.
  - `SCDS_FMSOCK_NA` — The address type (IPv4 or IPv6) does not apply to this `hostname`. If the `hostname` contains only one or more IPv4 mappings, the status of the second member in the array that is passed to this function is set to `SCDS_FMSOCK_NA`. The associated socket file descriptor is set to an unknown value, and should never be used.
  - `SCDS_FMSOCK_ERR` — The operation failed or timed out. The associated socket file descriptor is set to an unknown value, and should never be used.

PARAMETERS

The following parameters are supported:

handle	The handle that is returned by `scds_initialize(3HA)`.
socklist	An array of `SCDS_MAX_IPADDR_TYPES` members of type `scds_socket_t`. Each member in the array holds a status and a socket file descriptor for a TCP connection. This parameter is an output argument that is set by this function.
count	The number of members in the *socklist* array. Set this parameter to `SCDS_MAX_IPADDR_TYPES`.

addr	The `hostname`, TCP port number, and protocol identifier that specify where the process is listening.
timeout	The timeout value in seconds. Each socket gets the same time period for a connection to be established before it is timed out. As these time intervals proceed in parallel, this value is effectively the maximum time that the function takes to execute.

RETURN VALUES

The `scds_fm_net_connect()` function returns the following values:

`0`	The function succeeded. At least one socket connected.
`SCHA_ERR_INVAL`	The function was called with invalid paramaters.
Other nonzero values	Not a single connection could be established, due to a timeout, a refused connection, or some other error. You can inspect the `status` field of all members of the `socklist` array that are set to `SCDS_FMSOCK_ERR` to determine the exact error.

ERRORS

`SCHA_ERR_NOERR`	Indicates that the function succeeded.
`SCHA_ERR_INTERNAL`	Indicates that an internal error occurred while the function was executing.
`SCHA_ERR_STATE`	Indicates that the connection request was refused by the server.
`SCHA_ERR_TIMEOUT`	Indicates that the connection request timed out.

EXAMPLES

EXAMPLE 1 Using the `scds_fm_net_connect()` Function

```
/* this function is called repeatedly,
   after thorough_probe_interval seconds */
int probe(scds_handle_t scds_handle, ...)
{
    scds_socket_t socklist[SCDS_MAX_IPADDR_TYPES];
    ...

    /* for each hostname/port/proto */
    for (i = 0; i < netaddr->num_netaddrs, i++) {
        if (scds_fm_net_connect(scds_handle, socklist,
            SCDS_MAX_IPADDR_TYPES, netaddr[i], timeout) !=
            SCHA_ERR_NOERR)
            {
                /* failed completely */
                ...
        } else {
            /* at least one sock connected */
            for (j = 0, j < SCDS_MAX_IPADDR_TYPES, j++) {
                if (socklist[j].status == SCDS_FM_SOCK_NA)
                    continue;

                if (socklist[j].status == SCDS_FMSOCK_ERR) {
                    /* this particular connection failed */
                    scds_syslog(LOG_ERR, "Failed: %s",
                        scds_error_string(socklist[j].err));
```

EXAMPLE 1 Using the `scds_fm_net_connect()` Function *(Continued)*

```
                    continue;
                }

                /* use socklist[i].fd to perform write/read */
                ...
            }
            (void) scds_fm_net_disconnect(scds_handle, socklist,
                SCDS_MAX_IPADDR_TYPES, remaining_time);
        }

    }
    ...
    return (result);
}
```

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO

`scds_fm_net_disconnect`(3HA), `scds_fm_tcp_connect`(3HA), `scds_get_netaddr_list`(3HA), `scds_initialize`(3HA), `scha_calls`(3HA), `attributes`(5)

NAME

scds_fm_net_disconnect – terminate a TCP connection to an application

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_fm_net_disconnect(scds_handle_t handle,
     scds_socket_t *socklist, int count, time_t timeout);
```

DESCRIPTION

The `scds_fm_net_disconnect()` function terminates one or more TCP connections to a process that is being monitored.

An attempt is made to close all valid socket connections in the `socklist` array within the specified `timeout` interval. On return, each member of `socklist` contains the value `SCDS_FMSOCK_NA`.

PARAMETERS

The following parameters are supported:

handle	The handle that is returned by `scds_initialize(3HA)`.
socklist	The socket list that is returned by `scds_fm_net_connect(3HA)`. This argument is an input/output argument.
count	The number of members in the *socklist* array. Set this parameter to `SCDS_MAX_IPADDR_TYPES`.
timeout	The timeout value in seconds. Each socket gets the same time period to disconnect before it is timed out. As these time intervals proceed in parallel, this value is effectively the maximum time that the function takes to execute.

RETURN VALUES

The `scds_fm_net_disconnect()` function returns the following values:

`0`	The function succeeded.
`SCHA_ERR_INVAL`	The function was called with invalid paramaters.
Other nonzero values	The function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO | scds_fm_net_connect(3HA), scds_fm_tcp_disconnect(3HA), scds_initialize(3HA), scha_calls(3HA), attributes(5)

NAME

scds_fm_print_probes – print probe debugging information

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_fm_print_probes(scds_handle_t handle, int debug_level);
```

DESCRIPTION

The `scds_fm_print_probes()` function writes probe status information, reported with `scds_fm_action(3HA)`, to the system log. This information includes a list of all probe status history maintained by the DSDL and the timestamp associated with the probe status.

The DSDL defines the maximum debugging level, `SCDS_MAX_DEBUG_LEVEL`, as `9`.

If you specify a `debug_level` greater than the current debugging level being used, no information is written.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize(3HA)`.

debug_level Debugging level at which the data is to be written. It is an integer between `1` and `SCDS_MAX_DEBUG_LEVEL`, defined as `9` by the DSDL.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
include file

`/usr/cluster/lib/libdsdev.so`
library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_fm_action(3HA)`, `scds_initialize(3HA)`, `scds_syslog_debug(3HA)`, `attributes`(5)

NAME

scds_fm_sleep – wait for a message on a fault monitor control socket

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_fm_sleep(scds_handle_t handle, time_t timeout);
```

DESCRIPTION

The`scds_fm_sleep()` function waits for a data service application process tree that running under control of the process monitor facility to die. If no such death occurs within the specified timeout period, the function returns `SCHA_ERR_NOERR`.

If a data service application process tree death occurs, `scds_fm_sleep()` records `SCDS_COMPLETE_FAILURE` in the failure history and either restarts the process tree or fails it over according to the algorithm described in the `scds_fm_action(3HA)` man page. If a failover attempt is unsuccessful, a restart of the application is attempted.

If an attempted restart fails, the function returns `SCHA_ERR_INTERNAL`.

Note that if the failure history causes this function to do a failover, and the failover attempt succeeds, `scds_fm_sleep()` never returns.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize(3HA)`.

timeout The timeout period measured in seconds.

RETURN VALUES

The `scds_fm_sleep()` function returns the following:

`0` The function succeeded.

`non-zero` The function failed.

ERRORS

`SCHA_ERR_NOERR` Indicates the process tree has not died.

`SCHA_ERR_INTERNAL` Indicates the data service application process tree has died and failed to restart.

Other values Indicate the function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
include file

`/usr/cluster/lib/libdsdev.so`
library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO scha_calls(3HA), scds_fm_action(3HA), scds_initialize(3HA), attributes(5)

NAME

scds_fm_tcp_connect – establish a tcp connection to an application

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_fm_tcp_connect(scds_handle_t handle, int *sock,
     const char*hostname, int port, time_t timeout);
```

DESCRIPTION

The `scds_fm_tcp_connect()` function establishes a TCP connection with a process being monitored.

Retrieve the hostname with either `scds_get_rs_hostnames(3HA)` or `scds_get_rg_hostnames(3HA)`.

Consider using `scds_fm_net_connect(3HA)` instead of this function.

PARAMETERS

The following parameters are supported:

handle	The handle returned by `scds_initialize(3HA)`.
sock	A handle to the socket established by this function. This parameter is an output argument set by this function.
hostname	Name of the host where the process is listening. If the *hostname* maps to an IPv4 address only, or to both IPv4 and IPv6 addresses, this function uses the IPv4 mapping as the address at which to connect. If the *hostname* maps to an IPv6 address only, this function uses that IPv6 mapping as the address at which to connect.
port	TCP port number.
timeout	Timeout value in seconds.

RETURN VALUES

The `scds_fm_tcp_connect()` function returns the following:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Indicates the function succeeded.
`SCHA_ERR_STATE`	Indicates that an attempt to initiate a connection on a socket failed for reasons other than a timeout.
`SCHA_ERR_TIMEOUT`	Indicates the function timed out.
Other values	Indicate the function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Deprecated

SEE ALSO `scds_fm_net_connect`(3HA), `scds_fm_tcp_disconnect`(3HA), `scds_get_rg_hostnames`(3HA), `scds_get_rs_hostnames`(3HA), `scds_initialize`(3HA), `scha_calls`(3HA), `attributes`(5)

NAME

scds_fm_tcp_disconnect – terminate a tcp connection to an application

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_fm_tcp_disconnect(scds_handle_t handle, int sock,
    time_t timeout);
```

DESCRIPTION

The `scds_fm_tcp_disconnect()` function terminates a TCP connection with a process being monitored.

PARAMETERS

The following parameters are supported:

handle	The handle returned by `scds_initialize(3HA)`.
sock	The socket number returned by a previous call to `scds_fm_tcp_connect(3HA)`.
timeout	Timeout value in seconds.

RETURN VALUES

The following exit values are returned:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Indicates that the function succeeded.
`SCHA_ERR_TIMEOUT`	Indicates that the function timed out.
Other values	Indicate that the function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Deprecated

SEE ALSO

`scds_fm_net_disconnect(3HA)`, `scds_fm_tcp_connect(3HA)`, `scds_initialize(3HA)`, `scha_calls(3HA)`, `attributes`(5)

NAME

scds_fm_tcp_read – read data using a tcp connection to an application

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_fm_tcp_read(scds_handle_t handle, int sock, char
     *buffer, size_t *size, time_t timeout);
```

DESCRIPTION

The `scds_fm_tcp_read()` function reads data from a TCP connection with a process being monitored.

The *size* argument is an input and argument. On input, you specify the size of the buffer, bytes. On completion, the function places the data in *buffer* and specifies the actual number of bytes read in *size*. If the buffer is not big enough for the number of bytes read, the function returns a full buffer of *size* bytes, and you can call the function again for further data.

If the function times out, it returns `SCHA_ERR_TIMEOUT`. In this case, the function might return fewer bytes than requested, indicated by the value returned in *size*.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
sock	The socket number returned by a previous call to `scds_fm_tcp_connect(3HA)`
buffer	Data buffer
size	Data buffer size. On input, you specify the size of the buffer, in bytes. On output, the function returns the actual number of bytes read.
timeout	Timeout value in seconds.

RETURN VALUES

The `scds_fm_tcp_read()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Indicates the function succeeded.
`SCHA_ERR_TIMEOUT`	Indicates the function timed out.
Other values	Indicate the function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
include file

`/usr/cluster/lib/libdsdev.so`
library

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO `scds_fm_tcp_disconnect`(3HA), `scds_fm_tcp_write`(3HA), `scds_initialize`(3HA), `scha_calls`(3HA), `attributes`(5)

NAME

scds_fm_tcp_write – write data using a tcp connection to an application

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_fm_tcp_write(scds_handle_t handle, int sock, char
    *buffer, size_t *size, time_t timeout);
```

DESCRIPTION

The `scds_fm_tcp_write()` function writes data from by means of a TCP connection to a process being monitored.

The *size* argument is an input and output argument. On input, you specify the number of bytes to be written. On output, the function returns the number of bytes actually written. If the input and output values of *size* are not equal, an error has occurred. The function returns `SCHA_ERR_TIMEOUT` if it times out before writing all the requested data.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
sock	The socket number returned by a previous call to `scds_fm_tcp_connect(3HA)`
buffer	Data buffer
size	Data buffer size. On input, you specify the number of bytes to be written. On output, the function returns the number of bytes actually written.
timeout	Timeout value in seconds

RETURN VALUES

The `scds_fm_tcp_write()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Indicates the function succeeded.
`SCHA_ERR_TIMEOUT`	Indicates the function timed out.
Other values	Indicate the function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
include file

`/usr/cluster/lib/libdsdev.so`
library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO scds_fm_tcp_connect(3HA), scds_fm_tcp_read(3HA), scds_initialize(3HA), scha_calls(3HA), attributes(5)

NAME

scds_free_ext_property – free the resource extension property memory

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_free_ext_property(scha_ext_prop_value_t *property_value);
```

DESCRIPTION

The `scds_free_ext_property()` function reclaims memory allocated during calls to `scds_get_ext_property(3HA)`.

PARAMETERS

The following parameters are supported:

property_value Pointer to a property value

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_get_ext_property(3HA)`, `attributes`(5)

NAME

scds_free_netaddr_list – free the network address memory

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_free_netaddr_list(scds_netaddr_list_t *netaddr_list);
```

DESCRIPTION

The `scds_free_netaddr_list()` function reclaims memory allocated during calls to `scds_get_netaddr_list`(3HA). It deallocates the memory pointed to by *netaddr_list*.

PARAMETERS

The following parameters are supported:

netaddr_list Pointer to a list of hostname-port-protocol 3-tuples used by the resource group.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_get_netaddr_list`(3HA), `attributes`(5)

NAME

scds_free_net_list – free the network resource memory

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_free_net_list(scds_net_resource_list_t *net_resource_list);
```

DESCRIPTION

The `scds_free_net_list()` function reclaims memory allocated during calls to `scds_get_rg_hostnames`(3HA) or `scds_get_rs_hostnames`(3HA). It deallocates the memory pointed to by *netresource_list*.

PARAMETERS

The following parameters are supported:

netresource_list Pointer to a list of network resources used by the resource group

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_get_rg_hostnames`(3HA), `scds_get_rs_hostnames`(3HA), `attributes`(5)

NAME

scds_free_port_list – free the port list memory

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_free_port_list(scds_port_list_t *port_list);
```

DESCRIPTION

The `scds_free_port_list()` function reclaims memory allocated during calls to `scds_get_port_list(3HA)`. It deallocates the memory pointed to by *port_list*.

PARAMETERS

The following parameters are supported:

port_list Pointer to a list of port-protocol pairs used by the resource group

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_get_port_list(3HA)`, `attributes`(5)

NAME

scds_get_ext_property – retrieve an extension property

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_get_ext_property(scds_handle_t handle, const char
     *property_name, scha_prop_type_t property_type,
     scha_extprop_value_t **property_value);
```

DESCRIPTION

The `scds_get_ext_property()` function retrieves the value of a given extension property.

The name of the property is first looked up in the list of properties specified in the method argument list (`argv[]`, which was parsed by `scds_initialize()`). If the property name is not in the method argument list, it is retrieved using the Sun Cluster API. See `scha_calls(3HA)`.

Upon successful completion, the value of the property is placed in the appropriate variable in the union in a `scha_extprop_value_t` structure and a pointer to this structure is passed back to the caller in *property_value*.

You are responsible for freeing memory by using `scds_free_ext_property()`.

You can find information about the data types `scha_prop_type_t` and `scha_extprop_value_t` in `scha_calls(3HA)` and in the `<scha_types.h>` header file.

DSDL provides convenience functions to retrieve the values of some of the more commonly used resource extension properties. See the `scds_property_functions(3HA)` man page.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
property_name	Name of the property being retrieved
property_type	Property value type. Valid types are defined in `scha_calls(3HA)` and `property_attributes(5)`.
property_value	Pointer to a property value

RETURN VALUES

The `scds_get_ext_property()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_INVAL`	RTR file does not define the specified property.
`SCHA_ERR_NOERR`	Indicates the function succeeded.
Other values	Indicate the function failed. See `scha_calls(3HA)` for the meaning of the failure codes.

EXAMPLES

EXAMPLE 1 Using scds_get_ext_property

```
#include <scha_types.h>
#include <libdsdev.h>
#define INT_EXT_PROP "Int_extension_property"
...
int  retCode;
scha_extprop_value_t *intExtProp;
int retrievedValue;
...
    retCode = scds_get_ext_property(handle,
        INT_EXT_PROP, SCHA_PTYPE_INT, &intExtProp);
    if (retCode != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR,
            "Failed to retrieve the extension property %s: %s.",
            INT_EXT_PROP, scds_error_string(retCode));
       ...
    } else {
        retrievedValue = intExtProp->val.val_int;
        ...
        scds_free_ext_property(intExtProp);
        ...
    }
   ...
```

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_free_ext_property`(3HA), `scds_initialize`(3HA), `scds_property_functions`(3HA), `scha_calls`(3HA), `rt_reg`(4), `attributes`(5), `property_attributes`(5)

NOTES

Only the values of extension properties defined in the RTR file can be retrieved using this function. See `rt_reg`(4). If the extension property is not defined in the RTR file, `SCHA_ERR_INVAL` is returned.

NAME

scds_get_netaddr_list – get the network addresses used by a resource

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_get_netaddr_list(scds_handle_t handle,
     scds_netaddr_list_t **netaddr_list);
```

DESCRIPTION

The `scds_get_netaddr_list()` function returns all hostname, port, and protocol combinations that are in use by the resource. These combinations are derived by combining the `Port_list` property settings on the resource with all the hostnames in use by the resource, as returned by the `scds_get_rs_hostnames()` function.

Use `scds_get_netaddr_list()` in a fault monitor to monitor the resource, and to derive the list of hostnames, ports, and protocols that are in use by the resource .

Values for the protocol type are defined in header file `<rgm/libdsdev.h>`.

Free the memory that is allocated and returned by this function with `scds_free_netaddr_list()`.

PARAMETERS

The following parameters are supported:

handle The handle that is returned by `scds_initialize()`

netaddr_list The list of hostnames, ports, and protocols that are used by the resource group

RETURN VALUES

The `scds_get_netaddr_list()` function returns the following values:

`0` The function succeeded.

nonzero The function failed.

ERRORS

`SCHA_ERR_NOERR` Indicates that the function succeeded

Other values Indicate that the function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes.

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO

`scds_free_netaddr_list(3HA)`, `scds_get_rs_hostnames(3HA)`, `scha_calls(3HA)`, `r_properties(5)`, `attributes`(5)

NAME

scds_get_port_list – retrieve the port list used by a resource

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_get_port_list(scds_handle_t handle,
    scds_port_list_t **port_list);
```

DESCRIPTION

The `scds_get_port_list()` function returns a list of port-protocol pairs used by the resource. Values for the protocol type are defined in the header file `<netinet/in.h>`.

Free the memory allocated and returned by this function with `scds_free_port_list()`.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize()`
port_list	List of port-protocol pairs used by the resource group

RETURN VALUES

The `scds_get_port_list()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Indicates the function succeeded.
Other values	Indicate the function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/scha.h`
Include file

`/usr/cluster/lib/libscha.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_free_port_list(3HA)`, `scha_calls(3HA)`, `attributes`(5)

NAME

scds_get_resource_group_name – retrieve the resource group name

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

const char *scds_get_resource_group_name(scds_handle_t handle);
```

DESCRIPTION

The `scds_get_resource_group_name()` function returns a pointer to a character string that is the name of the resource group containing the resource passed to the calling program. The pointer is to memory belonging to the DSDL. Do not modify this memory. A call to `scds_close()` invalidates the pointer.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize()`

ERRORS

`NULL` Indicates an error condition such as not previously calling `scds_initialize(3HA)`

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/scha.h`
Include file

`/usr/cluster/lib/libscha.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_close(3HA)`, `scds_initialize(3HA)`, `scha_calls(3HA)`, `attributes`(5)

NAME

scds_get_resource_name – retrieve the resource name

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

const char *scds_get_resource_name(scds_handle_t handle);
```

DESCRIPTION

The `scds_get_resource_name()` function returns a pointer to a character string containing the name of the resource passed to the calling program. The pointer is to memory belonging to the DSDL. Do not modify this memory. A call to `scds_close()` invalidates the pointer.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize()`

ERRORS

`NULL` Indicates an error condition such as not previously calling `scds_initialize(3HA)`

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_close(3HA)`, `scds_initialize(3HA)`, `scha_calls(3HA)`, `attributes`(5)

NAME

scds_get_resource_type_name – retrieve the resource type name

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

const char *scds_get_resource_type_name(scds_handle_t handle);
```

DESCRIPTION

The `scds_get_resource_type_name()` function returns a pointer to a character string containing the name of the resource type of the resource passed to the calling program. The pointer is to memory belonging to the DSDL. Therefore, do not modify this memory. A call to `scds_close()` invalidates the pointer.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize()`

ERRORS

`NULL` Indicates an error condition such as not previously calling `scds_initialize()`

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_close(3HA)`, `scds_initialize(3HA)`, `scha_calls(3HA)`, `attributes`(5)

NAME

scds_get_rg_hostnames – get the network resources used in a resource group

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_get_rg_hostnames(char *resourcegroup_name,
     scds_net_resource_list_t **netresource_list);
```

DESCRIPTION

The `scds_get_rg_hostnames()` function retrieves a list of hostnames used by all the network resources in a resource group. This function returns a pointer to the list in *netresource_list*. It is possible for a resource group to contain no network resources or to contain resources that do not use network resources, so this function can return *netresource_list* set to `NULL`.

You can pass the name of any resource group name in the system to `scds_get_rg_hostnames()`. Use the hostnames returned by `scds_get_rg_hostnames()` to contact applications running in the specified resource group.

Free the memory allocated and returned by this function with `scds_free_net_list()`.

PARAMETERS

The following parameters are supported

resourcegroup_name	Name of the resource group for which data is to be retrieved
netresource_list	List of network resources used by the resource group

RETURN VALUES

The `scds_get_rg_hostnames()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls`(3HA) for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO scds_free_net_list(3HA), scds_get_rs_hostnames(3HA), scha_calls(3HA), attributes(5)

NAME

scds_get_rs_hostnames – get the network resources used by a resource

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_get_rs_hostnames(scds_handle_t handle,
     scds_net_resource_list_t **netresource_list);
```

DESCRIPTION

The `scds_get_rs_hostnames()` function retrieves a list of hostnames used by the resource. If the resource property `Network_resources_used` is set, then the hostnames correspond to the network resources listed in `Network_resources_used`. Otherwise, they correspond to all the network resources in the resource group containing the resource.

This function returns a pointer to the list in *netresource_list*. It is possible for a resource group to contain no network resources or to contain resources that do not use network resources, so this function can return *netresource_list* set to `NULL`.

Free the memory allocated and returned by this function with `scds_free_net_list(3HA)`.

PARAMETERS

The following parameters are supported

handle	The handle returned from `scds_initialize(3HA)`
netresource_list	List of network resources used by the resource group

RETURN VALUES

The `scds_get_rs_hostnames()` function returns the following:

`0`	The function succeeded
`non-zero`	The function failed

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_free_net_list(3HA)`, `scds_get_rg_hostnames(3HA)`, `scds_initialize(3HA)`, `scha_calls(3HA)`, `attributes`(5), `r_properties(5)`

NAME

scds_hasp_check – get status information about SUNW.HAStoragePlus resources used by a resource

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_hasp_check(scds_handle_t handle,
     scds_hasp_status_t *hasp_status);
```

DESCRIPTION

The `scds_hasp_check()` function retrieves status information about `SUNW.HAStoragePlus` resources used by a resource. This information is obtained from the state, online or otherwise, of all `SUNW.HAStoragePlus` resources that the resource depends upon using `Resource_dependencies` or `Resource_dependencies_weak` system properties defined for the resource.

Resource Type implementations can use `scds_hasp_check()` in `VALIDATE` and `MONITOR_CHECK` method callback implementation to ascertain whether checks specific to any filesystems that are managed by `SUNW.HAStoragePlus` resources, should be carried out or not.

When the function succeeds, a status code is stored in *hasp_status*. This code can be one of the following:

`SCDS_HASP_NO_RESOURCE`
: This indicates there is no `SUNW.HAStoragePlus` resource that this resource depends on.

`SCDS_HASP_ERR_CONFIG`
: Indicates that at least one of the `SUNW.HAStoragePlus` resource is in a different resource group then the current resource.

`SCDS_HASP_NOT_ONLINE`
: This indicates there is at least one `SUNW.HAStoragePlus` resource, that this resource depends on, which is not online on any potential primary node for this resource.

`SCDS_HASP_ONLINE_NOT_LOCAL`
: This indicates there is at least one `SUNW.HAStoragePlus` resource that this resource depends on, that is online on a different cluster node, that is, it. is not online on the cluster node where this function call is made.

`SCDS_HASP_ONLINE_LOCAL`
: This indicates that all `SUNW.HAStoragePlus` resources that this resource depends on are online on the node which called `scds_hasp_check()`.

These status codes have precedence over each other in the order in which they have been listed above. For example, if there is an `SUNW.HAStoragePlus` resource not online and another `SUNW.HAStoragePlus` resource online on a different node, the status code will be set to `SCDS_HASP_NOT_ONLINE` rather than `SCDS_HASP_ONLINE_NOT_LOCAL`.

All `SUNW.HAStoragePlus` resources who have their extension property `FilesystemMountPoints` set to empty, are ignored by `scds_hasp_check()`.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
hasp_status	Status of `SUNW.HAStoragePlus` resources used by the resource

RETURN VALUES

The `scds_hasp_check()` function returns the following:

`0`	The function succeeded
`non-zero`	The function failed

ERRORS

`SCHA_ERR_NOERR`	Indicates the function succeeded and the status code stored in `hasp_status` is valid
`SCHA_ERR_INTERNAL`	Indicates the function failed. Value stored in `hasp_status` is undefined and should be ignored.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_initialize(3HA)`, `attributes`(5)

NAME

scds_initialize – allocate and initialize DSDL environment

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_initialize(scds_handle_t *handleint argc, char
     *argv[]);
```

DESCRIPTION

The `scds_initialize()` function initializes the DSDL environment. You must call this function once at the beginning of each program or fault monitor that uses any other DSDL functions.

The `scds_initialize()` function does the following:

- Checks and processes the command line arguments (*argc* and *argv[]*) that the framework passes to the calling program and that must be passed along to `scds_initialize()`. No further processing of the command line arguments is required of the calling program. See EXAMPLES.
- Sets up internal data structures with information needed by the other functions in the DSDL. It retrieves resource, resource type, and resource group property values and stores them in these data structures. Values for any properties supplied on the command line by means of the *argv[]* argument take precedence over those retrieved from the RGM. That is, if a new value for a property has been specified in the command line arguments *(argv[]*) passed to the data service method, then this new value is returned by the function that retrieves that property's value. Otherwise, the existing value retrieved from the RGM is returned.
- Initializes the data service fault monitoring information
- Initializes the logging environment. All syslog messages are prefixed with: `SC[`*<resourceTypeName>*`,`*<resourceGroupName>*`,`*<resourceName>*`,`*<methodName>*

  Functions that send messages to syslog use the facility returned by `scha_cluster_getlogfacility()`. These messages can be forwarded to appropriate log files and users. See `syslog.conf`(4) for more information.
- Validates fault monitor probe settings. It verifies that the `Retry_interval` is greater than or equal to (`Thorough_probe_interval * Retry_count`). If this is not true, it sends an appropriate message to the syslog facility. You could call `scds_initialize()` and `scds_close()` in a `VALIDATE` method for this validation of the fault monitor probe settings even if you call no other DSDL functions in the `VALIDATE` method.

If `scds_initialize()` succeeds, you must call `scds_close()` before exiting the calling program.

If `scds_initialize()` fails, you must not call `scds_close()` to clean up. When `scds_initialize()` fails, do not call any other DSDL functions. They will return `SCHA_ERR_INVAL` or a `NULL` value. Rather, call `exit()` with a non-zero argument.

PARAMETERS

The following parameters are supported:

handle — A handle initialized by `scds_initialize()` and used by other DSDL functions

argc Number of arguments passed to the calling program

argv Pointer to an argument array passed to the calling program

RETURN VALUES

The `scds_initialize()` function returns the following:

`0` The function succeeded.

`non-zero` The function failed.

ERRORS

`SCHA_ERR_NOERR` Function succeeded

See `scha_calls(3HA)` for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using scds_initialize

```
int
main(int argc, char *argv[]){
 scds_handle_t handle;

 if (scds_initialize(&handle, argc, argv) !=
 SCHA_ERR_NOERR)
 exit(1);
 ...
 /* data service code */
...
 scds_close(&handle);
}
```

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_close(3HA)`, `scds_property_functions(3HA)`, `scha_calls(3HA)`, `scha_cluster_getlogfacility(3HA)`, `syslog.conf`(4), `r_properties(5)`

NAME

scds_pmf_get_status – determine if a PMF-monitored process tree exists

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_pmf_get_status(scds_handle_t handle,
     scds_pmf_type_t program_type, int instance, scds_pmf_status_t
     *pmf_status);
```

DESCRIPTION

The `scds_pmf_get_status()` function determines if the specified instance is being monitored under PMF control. This function is equivalent to the `pmfadm(1M)` command with the `-q` option.

PARAMETERS

The following parameters are supported:

handle — The handle returned from `scds_initialize()`

program_type — Type of program to execute. Valid types are:

`SCDS_PMF_TYPE_SVC`	Data service application
`SCDS_PMF_TYPE_MON`	Fault monitor
`SCDS_PMF_TYPE_OTHER`	Other

instance — For resources with multiple instances, this integer, starting at 0, uniquely identifies the instance. For single instance resources, use 0.

pmf_status — If PMF is monitoring the specified instance, *pmf_status* is set to `SCDS_PMF_MONITORED`. Otherwise it is set to `SCDS_PMF_NOT_MONITORED`.

RETURN VALUES

The `scds_pmf_get_status()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO pmfadm(1M), scds_initialize(3HA), scha_calls(3HA), attributes(5)

NAME

scds_pmf_restart_fm – restart fault monitor using PMF

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_pmf_restart_fm(scds_handle_t handle, int instance);
```

DESCRIPTION

The `scds_pmf_restart_fm()` function sends a `SIGKILL` signal to the fault monitor process tree to kill the fault monitor and then uses PMF to restart it. This function uses the `MONITOR_STOP_TIMEOUT` property as its timeout value. That is, `scds_pmf_restart_fm()` waits at most the value of the `MONITOR_STOP_TIMEOUT` property for the process tree to die.

If the `MONITOR_STOP_TIMEOUT` property is not explicitly set in the RTR file, the default timeout value is used.

One way to use this function is to call it in an `UPDATE` method to restart the monitor, possibly with new parameters.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize()`

instance For resources with multiple instances of the fault monitor, this integer, starting at 0, uniquely identifies the fault monitor instance. For single instance fault monitors, use 0.

RETURN VALUES

The `scds_pmf_restart_fm()` function returns the following:

`0` The function succeeded.

`non-zero` The function failed.

ERRORS

`SCHA_ERR_NOERR` Function succeeded

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`pmfadm(1M)`, `scha_calls(3HA)`, `signal`(3HEAD), `attributes`(5) , `r_properties(5)`

NAME

scds_pmf_signal – send a signal to a process tree under PMF control

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_pmf_signal(scds_handle_t handle, scds_pmf_type_t
     program_type, int instance, int signal, time_t timeout);
```

DESCRIPTION

The `scds_pmf_signal()` function sends the specified signal to a process tree running under PMF control. This function is equivalent to the `pmfadm(1M)` command with the `-k` option.

After sending the signal, the `scds_pmf_signal()` function waits for the specified timeout period for the process tree to die, before returning. A value of `0` for `timeout` tells the function to return immediately without waiting for any process to exit. A value of `-1` tells the function to wait indefintely for the processes to exit.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize()`	
program_type	Type of program to execute. Valid types are:	
	`SCDS_PMF_TYPE_SVC`	Data service application
	`SCDS_PMF_TYPE_MON`	Fault monitor
	`SCDS_PMF_TYPE_OTHER`	Other
instance	For resources with multiple instances, this integer, starting at 0, uniquely identifies the instance. For single instance resources, use 0.	
signal	Solaris signal to send. See `signal`(3HEAD).	
timeout	Timeout period in seconds.	

RETURN VALUES

The `scds_pmf_signal()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_TIMEOUT`	The process tree did not exit within the specified timeout period after the signal was sent.
`SCHA_ERR_NOERR`	The function succeeded.
Other values	Indicate the function failed. See `scha_calls(3HA)` for the meaning of failure codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO `pmfadm`(1M), `scds_initialize`(3HA), `scha_calls`(3HA), `signal`(3HEAD), `attributes`(5)

NAME

scds_pmf_start – execute a program under PMF control

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_pmf_start(scds_handle_t handle, scds_pmf_type_t
     program_type, int instance, const char *command, int
     child_monitor_level);
```

DESCRIPTION

The `scds_pmf_start()` function executes a program, specified by *command*, under PMF control. This function is equivalent to the `pmfadm(1M)` command with the `-c` option.

The *command* argument contains a command line and command line arguments that are passed to the function.

When you start a data service application or other process (program type `SCDS_PMF_TYPE_SVC` or `SCDS_PMF_TYPE_OTHER`) under PMF with `scds_pmf_start()`, you choose the level of child processes to monitor by using the `child_monitor_level` argument. Values for the `child_monitor_level` argument are `none`, `some` or `all`. The `child_monitor_level` argument specifies that children up to and including level `child_monitor_level` will be monitored. The original process is executed at level 0, its children at level 1, their children at level 2, and so on. Any new fork operation produces a new level of children. Specify `-1` to monitor all levels of children.

For example, if the command to start is a daemon, the appropriate `child_monitor_level` is `1`. If the command to start is a script that starts a daemon, the appropriate value for `child_monitor_level` is `2`.

For a fault monitor (program type `SCDS_PMF_TYPE_MON`), the `child_monitor_level` argument is ignored and `0` is used.

If the underlying application process is already running, `scds_pmf_start()` prints a `syslog()` error and returns `SCHA_ERR_INTERNAL` because the RGM guarantees that two calls to a `START` function on a node must have an intervening `STOP` function.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`	
program_type	Type of program to execute. Valid types are:	
	`SCDS_PMF_TYPE_SVC`	Data service application
	`SCDS_PMF_TYPE_MON`	Fault monitor
	`SCDS_PMF_TYPE_OTHER`	Other
instance	For resources with multiple instances, this integer, starting at 0, uniquely identifies the instance. For single instance resources, use 0.	

command Command, including command line arguments, to execute under PMF control.

child_monitor_level For `program_type SCDS_PMF_TYPE_SVC` and `SCDS_PMF_TYPE_OTHER`, this argument specifies the level of child processes to be monitored (equivalent to the `-C` option to `pmfadm`). Use `-1` to specify all levels of child processes. For `program_type SCDS_PMF_TYPE_MON`, this argument is ignored.

RETURN VALUES

The `scds_pmf_start()` function returns the following:

`0` The function succeeded.

`non-zero` The function failed.

ERRORS

`SCHA_ERR_INTERNAL` The underlying application process is already running.

`SCHA_ERR_NOERR` The function succeeded.

Other values The function failed. See `scha_calls`(3HA) for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`pmfadm`(1M), `scds_initialize`(3HA), `scds_pmf_stop`(3HA), `scds_svc_wait`(3HA), `scha_calls`(3HA), `attributes`(5)

NAME

scds_pmf_stop – terminate a process that is running under PMF control

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_pmf_stop(scds_handle_t handle, scds_pmf_type_t
    program_type, int instance, int signal, time_t timeout);
```

DESCRIPTION

The `scds_pmf_stop()` function stops a program that is running under PMF control. It is equivalent to the `pmfadm(1M)` command with the `-s` option.

If the requested instance is not running, `scds_pmf_stop()` returns with value `SCHA_ERR_NOERR`.

If the requested instance is running, then the specified signal is sent to the instance. If the instance fails to die within a period of time equal to 80% of the timeout value, `SIGKILL` is sent to the instance. If the instance then fails to die within a period of time equal to 15% of the timeout value, the function is considered to have failed and returns `SCHA_ERR_TIMEOUT`. The remaining 5% of the timeout argument is presumed to have been absorbed by this function's overhead.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`	
program_type	Type of program to execute. Valid types are:	
	`SCDS_PMF_TYPE_SVC`	Data service application
	`SCDS_PMF_TYPE_MON`	Fault monitor
	`SCDS_PMF_TYPE_OTHER`	Other
instance	For resources with multiple instances, this integer, starting at 0, uniquely identifies the instance. For single instance resources, use 0.	
signal	Solaris signal to send kill the instance. See `signal`(3HEAD). Use `SIGKILL` if the specified signal fails to kill the instance.	
timeout	Timeout period measured in seconds.	

RETURN VALUES

The `scds_pmf_stop()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_TIMEOUT`	The function timed out.
`SCHA_ERR_NOERR`	The function succeeded.
Other values	Indicate the function failed. See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`pmfadm`(1M), `scds_initialize`(3HA), `scds_pmf_start`(3HA), `scha_calls`(3HA), `signal`(3HEAD), `attributes`(5)

NAME

scds_pmf_stop_monitoring – stop monitoring a process that is running under PMF control

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_pmf_stop_monitoring(scds_handle_t handle,
     scds_pmf_type_t program_type, int instance);
```

DESCRIPTION

The `scds_pmf_stop_monitoring()` function stops the monitoring of a process tree that is running under PMF control. PMF does not send a signal to stop the process. Rather, PMF makes no future attempts to restart the process.

If the requested process is not under PMF control, `scds_pmf_stop_monitoring()` returns, with value `SCHA_ERR_NOERR`.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize(3HA)`

program_type Type of program to execute. Valid types are:

`SCDS_PMF_TYPE_SVC`	Data service application
`SCDS_PMF_TYPE_MON`	Fault monitor
`SCDS_PMF_TYPE_OTHER`	Other

instance For resources with multiple instances, this integer, starting at 0, uniquely identifies the instance. For single instance resources, use 0.

RETURN VALUES

The `scds_pmf_stop_monitoring()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	The function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO pmfadm(1M), scds_initialize(3HA), scds_pmf_start(3HA), scds_pmf_stop(3HA), scha_calls(3HA), attributes(5)

NAME

scds_print_netaddr_list – print the contents of a list of hostname-port-protocol 3-tuples used by a resource group

SYNOPSIS

```
cc [flags...] -I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_print_netaddr_list(scds_handle_t handle, int debug_level,
     const scds_netaddr_list_t *netaddr_list);
```

DESCRIPTION

The `scds_print_netaddr_list()` function writes the contents of a list of hostname-port-protocol 3-tuples, pointed to by *netaddr_list*, to the system log, at the debugging level specified by *debug_level*. If the specified debugging level is greater than the debugging level currently being used, no information is written.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
debug_level	The debugging level at which the data is to be written
netaddr_list	Pointer to a list of hostname-port-protocol 3-tuples used by the resource group, retrieved with `scds_get_netaddr_list(3HA)`

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_get_netaddr_list(3HA)`, `scds_initialize(3HA)`, `scds_syslog_debug(3HA)`, `attributes`(5)

NAME

scds_print_net_list – print the contents of a network resource list

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

voidscds_print_net_list(scds_handle_t handle, int debug_level, const
     scds_net_resource_list_t *netresource_list);
```

DESCRIPTION

The `scds_print_net_list()` function writes the contents of the network resource list, pointed to by *netresource_list*, to the system log, at the debugging level specified by *debug_level*. If the specified debugging level is greater than the debugging level currently being used, no information is written.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
debug_level	Debugging level at which the data is to be written
netresource_list	Pointer to an initialized network resource list, retrieved with either `scds_get_rg_hostnames(3HA)` or `scds_get_rs_hostnames(3HA)`

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_get_rg_hostnames(3HA)`, `scds_get_rs_hostnames(3HA)`, `scds_initialize(3HA)`, `scds_syslog_debug(3HA)`, `attributes`(5)

NAME

scds_print_port_list – print the contents of a port list

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_print_port_list(scds_handle_t handle, int debug_level,
     const scds_port_list_t *port_list);
```

DESCRIPTION

The `scds_print_port_list()` function writes the contents of a port list, pointed to by *port_list*, to the system log, at the debugging level specified by *debug_level*. If the specified debugging level is greater than the debugging level currently being used, no information is written.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
debug_level	Debugging level at which the data is to be written
port_list	Pointer to a list of port-protocol pairs used by the resource group, retrieved with `scds_get_port_list()`.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_get_port_list(3HA)`, `scds_initialize(3HA)`, `scds_syslog_debug(3HA)`, `attributes`(5)

NAME

scds_property_functions – A set of convenience functions to retrieve values of commonly used resource properties, resource group properties, resource type properties, and extension properties

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

return_value scds_get_property_name(scds_handle_t handle);
```

DESCRIPTION

The DSDL provides a set of convenience functions to retrieve values of commonly used resource properties, resource group properties, resource type properties, and extension properties. Retrieve user-defined extension properties with scds_get_ext_property(3HA).

All convenience functions use the following conventions:

- The functions take only the *handle* argument.
- Each function corresponds to a particular property.
- The return value type of the function matches the type of the property value it retrieves.
- These functions do not return errors because the return values have been pre-computed in scds_initialize(3HA). For functions that return pointers, a NULL value is returned when an error condition is encountered, for example, when scds_initialize() was not previously called.
- If a new value for a property has been specified in the command-line arguments passed to the calling program (*argv[]*), this new value is returned. Otherwise, these functions return the value retrieved from the RGM.
- Some of these convenience functions return a pointer to memory belonging to the DSDL. Do not modify this memory. A call to scds_close(3HA) invalidates this pointer.

See the r_properties(5), rg_properties(5), and rt_properties(5) man pages for descriptions of standard properties. See the individual data service man pages for descriptions of extension properties.

See the scha_calls(3HA) man page and the <scha_types.h> header file for information about the data types used by these functions, such as scha_prop_type_t, scha_extprop_value_t, scha_initnodes_flag_t, scha_str_array_t, scha_failover_mode_t, scha_switch_t, and scha_rsstatus_t.

These functions use the following naming conventions:

Resource property
  scds_get_rs_*property-name*

Resource group property
  scds_get_rg_*property-name*

Resource type property
  scds_get_rt_*property-name*

Commonly used extension property
`scds_get_ext_`*property-name*

Note – Property names are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify property names.

Resource-Specific Functions

The function declaration returns values for the resource property to retrieve. Some of the properties' values are explicitly set either in the RTR file or by a `scrgadm(1M)` command. Others are determined dynamically by the RGM. The functions return data types appropriate for the requested property.

```
Cheap_probe_interval
  int scds_get_rs_cheap_probe_interval(scds_handle_t handle)

Failover_mode
  scha_failover_mode_t scds_get_rs_failover_mode(scds_handle_t
  handle)

Monitor_stop_timeout
  int scds_get_rs_monitor_stop_timeout(scds_handle_t handle)

Monitored_switch
  scha_switch_t scds_get_rs_monitored_switch(scds_handle_t
  handle)

Network_resources_used
  scha_str_array_t *
  scds_get_rs_network_resources_used(scds_handle_t handle)

On_off_switch
  scha_switch_t scds_get_rs_on_off_switch(scds_handle_t handle)

Resource_dependencies
  const scha_str_array_t *
  scds_get_rs_resource_dependencies(scds_handle_t handle)

Resource_dependencies_restart
  const scha_str_array_t *
  scds_get_rs_resource_dependencies_restart(scds_handle_t
  handle)

Resource_dependencies_weak
  const scha_str_array_t *
  scds_get_rs_resource_dependencies_weak(scds_handle_t handle)

Resource_project_name
  const char * scds_get_rs_resource_project_name(scds_handle_t
  handle)

Retry_count
  int scds_get_rs_retry_count(scds_handle_t handle)

Retry_interval
  int scds_get_rs_retry_interval(scds_handle_t handle)
```

```
Scalable
  boolean scds_get_rs_scalable(scds_handle_t handle)

Start_timeout
  int scds_get_rs_start_timeout(scds_handle_t handle)

Stop_timeout
  int scds_get_rs_stop_timeout(scds_handle_t handle)

Thorough_probe_interval
  int scds_get_rs_thorough_probe_interval(scds_handle_t handle)
```

Resource Group-Specific Functions

The function declaration returns values for the resource group property to retrieve. Some of the properties' values are explicitly set either in the RTR file or by a scrgadm(1M) command. Others are determined dynamically by the RGM. The functions return data types appropriate for the requested property.

```
Desired_primaries
  int scds_get_rg_desired_primaries(scds_handle_t handle)

Global_resources_used
  const scha_str_array_t *
  scds_get_rg_global_resources_used(scds_handle_t handle)

Implicit_network_dependencies
  boolean_t
  scds_get_rg_implicit_network_dependencies(scds_handle_t
  handle)

Maximum_primaries
  int scds_get_rg_maximum_primaries(scds_handle_t handle)

Nodelist
  const scha_str_array_t * scds_get_rg_nodelist (scds_handle_t
  handle)

Pathprefix
  const char * scds_get_rg_pathprefix(scds_handle_t handle)

Pingpong_interval
  int scds_get_rg_pingpong_interval(scds_handle_t handle)

Resource_list
  const scha_str_array_t *
  scds_get_rg_resource_list(scds_handle_t handle)

RG_affinities
  const scha_str_array_t *
  scds_get_rg_rg_affinities(scds_handle_t handle)

RG_mode
  scha_rgmode_t scds_get_rg_rg_mode(scds_handle_t handle)

RG_project_name
  const char * scds_get_rg_rg_project_name(scds_handle_t handle)
```

Resource Type-Specific Functions

The function declaration returns values for the resource type property to retrieve. Some of the properties' values are explicitly set either in the RTR file or by a scrgadm(1M) command. Others are determined dynamically by the RGM. The functions return data types appropriate for the requested property.

```
API_version
  int scds_get_rt_api_version(scds_handle_t handle)

Failover
  boolean_t scds_get_rt_failover(scds_handle_t handle)

Init_nodes
  scha_initnodes_flag_t scds_get_rt_init_nodes(scds_handle_t
  handle)

Installed_nodes
  const scha_str_array_t *
  scds_get_rt_installed_nodes(scds_handle_t handle)

RT_basedir
  const char * scds_get_rt_rt_basedir(scds_handle_t handle)

RT_version
  const char * scds_get_rt_rt_version(scds_handle_t handle)

Single_instance
  boolean_t scds_get_rt_single_instance(scds_handle_t handle)

Start_method
  const char * scds_get_rt_start_method(scds_handle_t handle)

Stop_method
  const char * scds_get_rt_stop_method(scds_handle_t handle)
```

Extension Property-Specific Functions

The function declaration returns values for the resource extension property to retrieve. Some of the properties' values are explicitly set either in the RTR file or by a scrgadm(1M) command. The functions return data types appropriate for the requested property.

A resource type can define extension properties beyond the four listed here, but these four properties have convenience functions defined for them. You retrieve these properties with these convenience functions or with the scds_get_ext_property(3HA) function. You must use `scds_get_ext_property()` to retrieve extension properties other than these four.

```
Confdir_list
  scha_str_array_t * scds_get_ext_confdir_list(scds_handle_t
  handle)

Monitor_retry_count
  int scds_get_ext_monitor_retry_count(scds_handle_t handle)

Monitor_retry_interval
  int scds_get_ext_monitor_retry_interval(scds_handle_t handle)
```

```
Probe_timeout
  int scds_get_ext_probe_timeout(scds_handle_t handle)
```

PARAMETERS

The following parameter is supported for all the convenience functions:

handle The handle that is returned from `scds_initialize(3HA)`.

RETURN VALUES

Each function returns a value type that matches the type of the property value it retrieves.

These functions do not return errors because the return values have been pre-computed in `scds_initialize(3HA)`. For functions that return pointers, a `NULL` value is returned when an error condition is encountered, for example, when `scds_initialize()` was not previously called

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scrgadm(1M)`, `scds_close(3HA)`, `scds_get_ext_property(3HA)`, `scds_get_port_list(3HA)`, `scds_get_resource_name(3HA)`, `scds_get_resource_group_name(3HA)`, `scds_get_resource_type_name(3HA)`, `scds_initialize(3HA)`, `scha_calls(3HA)`, `attributes`(5), `r_properties(5)`, `rg_properties(5)`, and `rt_properties(5)`

NAME

scds_restart_resource – restart a resource

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_restart_resource(scds_handle_t handle);
```

DESCRIPTION

The `scds_restart_resource()` function provides resource-level granularity for the restart operation. This function calls the `STOP` method and then the `START` method for the resource passed to the calling program. If `PRENET_START` and `POSTNET_STOP` methods are defined for the resource type, they are ignored. Call this function from the fault monitor.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize(3HA)`

RETURN VALUES

The `scha_restart_resource()` function returns the following:

`0` The function succeeded.

`non-zero` The function failed.

ERRORS

`SCHA_ERR_NOERR` Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`rt_callbacks(1HA)`, `scds_restart_rg(3HA)`, `scha_calls(3HA)`, `scha_control(3HA)`, `attributes`(5)

NAME

scds_restart_rg – restart a resource group

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_restart_rg(scds_handle_t handle);
```

DESCRIPTION

The `scds_restart_rg()` function performs an `scha_control(3HA)` `SCHA_RESTART` operation on the resource group containing the resource passed to the calling program. Call this function from the fault monitor.

When this function succeeds, it does not return. Therefore, treat this function as the last piece of code to be executed in the calling program.

PARAMETERS

The following parameters are supported:

handle The handle returned from `scds_initialize(3HA)`

RETURN VALUES

The `scds_restart_rg()` function returns the following:

`0` The function succeeded.

`non-zero` The function failed.

ERRORS

`SCHA_ERR_NOERR` Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scha_calls(3HA)`, `scha_control(3HA)`, `scds_initialize(3HA)`, `scds_restart_resource(3HA)`, `attributes`(5)

NAME

scds_simple_net_probe – probe by establishing and terminating a TCP connection to an application

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_simple_net_probe(scds_handle_t handle,
    scds_netaddr_t addr, time_t timeout, scds_fmsock_status_t
    *status, int count);
```

DESCRIPTION

The `scds_simple_net_probe()` function is a wrapper function around `scds_fm_net_connect(3HA)` and `scds_fm_net_disconnect(3HA)`. For hosts that have multiple mappings, `scds_simple_net_probe()` handles both IPv4 and IPv6 addresses for the supplied `hostname`.

You can retrieve a list of network addresses for the resource by using `scds_get_netaddr_list(3HA)`.

The status for a connect to, or disconnect from, an IPv4 target is stored in the first member of the `scds_fmsock_status_t` array. The second member contains the status for an IPv6 target. If the `hostname` that is supplied to this function does not contain an IPv4 or IPv6 mapping, the corresponding status is set to `SCDS_FMSOCK_NA`.

PARAMETERS

The following parameters are supported:

handle	The handle returned by `scds_initialize(3HA)`.
addr	The `hostname`, TCP port number, and protocol identifier that specify where the process is listening.
timeout	The timeout value in seconds to wait for a successful connection. Each socket (IPv4 or IPv6) gets the same timeout period, and timeouts proceed in parallel.
status	Array of `SCDS_MAX_IPADDR_TYPES` members of type `scds_fmsock_status_t`. Each member in the array holds a status. This parameter is an output argument that is set by this function.
count	The number of members in the *socklist* array. Set this parameter to `SCDS_MAX_IPADDR_TYPES`.

RETURN VALUES

The `scds_simple_net_probe()` function returns the following values:

`0`	The function succeeded.
`SCHA_ERR_INVAL`	The function was called with invalid paramaters.
Other nonzero values	At least one connect operation failed due to a timeout, a refused connection, or some other error. Inspect the `err` field of all members of the `socklist` array that are set to `SCDS_FMSOCK_ERR` to determine the exact error.

nonzero — At least one connect or disconnect operation failed. You can inspect the `scds_fmsock_status_t` array to determine if the failure was in an IPv4 target, an IPv6 target, or both.

ERRORS

`SCHA_ERR_NOERR` — Indicates that the function succeeded.

`SCHA_ERR_INTERNAL` — Indicates that an internal error occurred while the function was executing.

`SCHA_ERR_STATE` — Indicates that the connection request was refused by the server.

`SCHA_ERR_TIMEOUT` — Indicates that the connection request timed out.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO

`scds_fm_net_connect`(3HA), `scds_fm_net_disconnect`(3HA), `scds_get_netaddr_list`(3HA), `scds_initialize`(3HA), `scds_simple_probe`(3HA), `scha_calls`(3HA), `attributes`(5)

NAME

scds_simple_probe – probe by establishing and terminating a TCP connection to an application

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_simple_probe(scds_handle_t handle, const char
    *hostname, int port, time_t timeout);
```

DESCRIPTION

The `scds_simple_probe()` function is a wrapper function around `connect`(3SOCKET) and `close`(2) to run under a timeout.

Retrieve the *hostname* with either `scds_get_rg_hostnames`(3HA) or `scds_get_rs_hostnames`(3HA).

Consider using `scds_simple_net_probe`(3HA) instead of this function.

PARAMETERS

The following parameters are supported:

handle	The handle returned by `scds_initialize`(3HA).
hostname	Internet hostname of the machine to which to connect.
port	Port number with which to make the connection.
timeout	Timeout value in seconds (to wait for a successful connection).

RETURN VALUES

The `scds_simple_probe()` function returns the following:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Indicates that the function succeeded.
`SCHA_ERR_TIMEOUT`	Indicates that the function timed out.

See `scha_calls`(3HA) for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Deprecated

SEE ALSO close(2), connect(3SOCKET), scds_fm_net_connect(3HA), scds_fm_net_disconnect(3HA), scds_get_rg_hostnames(3HA), scds_get_rs_hostnames(3HA), scds_initialize(3HA), scds_simple_net_probe(3HA), scha_calls(3HA), attributes(5)

NAME

scds_svc_wait – wait for the specified timeout period for a monitored process to die

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_svc_wait(scds_handle_t handle, time_t timeout);
```

DESCRIPTION

The `scds_svc_wait()` function waits for the specified timeout period for a monitored process group to die. It waits upon all process groups started by `scds_pmf_start(3HA)` for the resource passed to the calling `START` method. The `scds_svc_wait()` function uses the `Retry_interval` and `Retry_count` properties of the resource to limit the number of process deaths to wait on. If the number of process deaths during `Retry_interval` reaches the value of `Retry_count`, `scds_svc_wait()` returns with `SCHA_ERR_FAIL`.

If the number of process failures is below the value of `Retry_count`, the process is restarted and `scds_svc_wait()` waits the full timeout period for further process deaths. The counting of process failures spans successive calls to `scds_svc_wait()`.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
timeout	Timeout period measured in seconds

RETURN VALUES

The `scds_svc_wait()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_TIMEOUT`	The function timed out.
`SCHA_ERR_NOERR`	No process deaths occurred, or a process was successfully restarted.
`SCHA_ERR_FAIL`	The number of failures reached the value of the `Retry_count` property.
`SCHA_ERR_STATE`	A system error or an otherwise unexpected error occurred.

See `scha_calls(3HA)` for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using `scds_svc_wait()` in a `START` Method

The following example shows how you could use `scds_svc_wait` in a `START` method to return early if the service fails to start. After starting an application process with `scds_pmf_start()`, a `START` method must wait for the application to fully initialize itself and become available before returning success. If the application fails to start, the `START` method must wait the entire `Start_timeout` period before returning with failure. Using `scds_svc_wait()`, as in the following example, allows `START` methods to restart applications up to `Retry_count` times and return early with failure from the `START` method if the service is unable to start up.

EXAMPLE 1 Using `scds_svc_wait()` in a `START` Method *(Continued)*

```
/*
 * scds_svc_wait is a subroutine in a START method to
 * check that the service is fully available before returning.
 * Calls svc_probe() to check service availability.
 */
int
svc_wait(scds_handle_t handle)
{
    while (1) {
        /* Wait for 5 seconds */
        if (scds_svc_wait(handle, 5) != SCHA_ERR_NOERR) {
            scds_syslog(LOG_ERR, "Service failed to start.");
            return (1);        /* Start Failure */
        }
        /* Check if service is fully up every 5 seconds */
        if (svc_probe(handle) == 0) {
            scds_syslog(LOG_INFO, "Service started successfully.");
            return (0);
        }
    }
    return (0);
}
```

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scds_initialize`(3HA), `scds_pmf_start`(3HA) , `scha_calls`(3HA), `attributes`(5), `r_properties`(5)

NOTES

- If the `START` method exceeds the `Start_timeout` setting on the resource, the RGM will kill the `START` method even if the `START` method is currently waiting for `scds_svc_wait()` to return.
- If `Retry_interval` on the resource is larger then `Start_timeout`, the `START` method could be timed out by the RGM even if the number of failures is below `Retry_count`.
- If a `START` method starts multiple process groups with multiple calls to `scds_pmf_start()`, `scds_svc_wait()` starts process groups as they die. It does not enforce any dependencies between process groups. Do not use `scds_svc_wait()` if there is a dependency between process groups such that

failure of one process group requires a restart of other process groups. Instead, use `sleep()` to wait between health checks of the process groups.

NAME

scds_syslog – write a message to the system log

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_syslog(int priority, const char *format...);
```

DESCRIPTION

The `scds_syslog()` function writes a message to the system log. It uses the facility returned by the`scha_cluster_getlogfacility(3HA)` function. You can forward these messages to appropriate log files and users. See `syslog.conf`(4) for more information.

All syslog messages are prefixed with:
`SC[`<*resourceTypeName*>`,`<*resourceGroupName*>`,`<*resourceName*>`,`<*methodName*>

Caution – Messages written to the system log are not internationalized. Do not use `gettext()` or other message translation functions in conjunction with this function.

PARAMETERS

The following parameters are supported:

priority	Message priority, as specified by `syslog`(3C)
format	Message format string, as specified by `printf`(3C)
...	Variables, indicated by the *format* parameter, as specified by `printf()`

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`printf`(3C), `scds_syslog_debug(3HA)`, `scha_cluster_getlogfacility(3HA)`, `syslog`(3C), `syslog.conf`(4), `attributes`(5)

NAME

scds_syslog_debug – write a debugging message to the system log

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

void scds_syslog_debug(int debug_level, const char *format...);
```

DESCRIPTION

The `scds_syslog_debug()` function writes a debugging message to the system log. It uses the facility returned by the `scha_cluster_getlogfacility(3HA)` function.

All syslog messages are prefixed with:
`SC[`<*resourceTypeName*>`,`<*resourceGroupName*>`,`<*resourceName*>`,`<*methodName*>

If you specify a *debug_level* greater than the current debugging level being used, no information is written.

The DSDL defines the maximum debugging level, `SCDS_MAX_DEBUG_LEVEL`, as `9`. The `scds_initialize(3HA)` function, which the calling program must call before `scds_syslog_debug()`, retrieves the current debugging level from the file: `/var/cluster/rgm/rt/`<*resourceTypeName*>`/loglevel`.

Caution – Messages written to the system log are not internationalized. Do not use `gettext()` or other message translation functions in conjunction with this function.

PARAMETERS

The following parameters are supported:

debug_level	Debugging level at which this message is to be written. Valid debugging levels are between 1 and `SCDS_MAX_DEBUG_LEVEL`, which is defined as 9 by the DSDL. If the specified debugging level is greater than the debugging level set by the calling program, the message is not written to the system log.
format	Message format string, as specified by `printf`(3C)
...	Variables, indicated by the *format* parameter, as specified by `printf`(3C)

EXAMPLES

EXAMPLE 1 Display All Debugging Messages

To see all debugging messages for resource type `SUNW.iws`, issue the following command on all nodes of your cluster

```
echo 9 > /var/cluster/rgm/rt/SUNW.iws/loglevel
```

EXAMPLE 2 Suppress Debugging Messages

To suppress debugging messages for resource type `SUNW.iws`, issue the following command on all nodes of your cluster

```
echo 0 > /var/cluster/rgm/rt/SUNW.iws/loglevel
```

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`printf`(3C), `scds_syslog`(3HA), `scha_cluster_getlogfacility`(3HA), `syslog`(3C), `syslog.conf`(4), `attributes`(5)

NAME

scds_timerun – execute a given command in a given amount of time

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l dsdev
#include <rgm/libdsdev.h>

scha_err_t scds_timerun(scds_handle_t handle, const char *command,
    time_t timeout, int signal, int *cmd_exit_code);
```

DESCRIPTION

The `scds_timerun()` function executes a specified command using `hatimerun(1M)`. If the command does not complete within the allotted time period,which is specified by the *timeout* argument, `scds_timerun()` sends a signal, specified by the *signal* argument, to kill it.

The *command* argument does not support I/O redirection. However, you can write a script to perform redirection and then identify this script in the *command* argument as the command for `scds_timerun()` to execute.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize(3HA)`
command	String containing the command to run
timeout	Time, in seconds, allotted to run the command
signal	Signal to kill the command if it is still running when the timeout expires. If `signal = -1`, then `SIGKILL` is used. See `signal`(3HEAD).
cmd_exit_code	Return code from execution of the command

RETURN VALUES

The `scds_timerun()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	The command executed and `cmd_exit_code` contains the child program's exit status.
`SCHA_ERR_INTERNAL`	The timeout did not occur, but some other error was detected by `scds_timerun()` that was not an error detected by the child program. Or `hatimerun(1M)` caught the signal `SIGTERM`.
`SCHA_ERR_INVAL`	There was an invalid input argument.
`SCHA_ERR_TIMEOUT`	The timeout occurred before the command specified by the *command* argument finished executing.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/rgm/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`hatimerun`(1M), `scds_initialize`(3HA), `scha_calls`(3HA), `signal`(3HEAD), `attributes`(5)

NAME

scha_calls – Sun Cluster library functions used in the implementation of callback methods and monitors of resource types

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_get_function(handle, const char *tag...);

scha_err_t scha_control(const char *tag...);
```

DESCRIPTION

The Sun Cluster library functions `scha_resource_get(3HA)`, `scha_resourcetype_get(3HA)`, `scha_resourcegroup_get(3HA)`, `scha_cluster_get(3HA)`, `scha_control(3HA)`, `scha_strerror(3HA)`, and `scha_resource_setstatus(3HA)` provide an interface to be used in the implementation of callback methods and monitors of resource types. The resource types represent services that are controlled by the cluster's Resource Group Manager (RGM) facility.

The "get" functions access cluster configuration information. All these functions have the same general signature. These functions take a *handle* argument that is returned from a previous call to an "open" function. This *handle* indicates the object in the cluster configuration that is to be accessed. A *tag* argument indicates the property of the object that is to be accessed. The value of *tag* determines whether additional arguments are needed and the type of a final "out" argument through which the requested information is returned. You can make repeated "get" calls with the same handle until a "close" call, which invalidates the handle and frees memory that is allocated for values that are returned from the "get" calls.

Memory, if needed to return a value, is allocated for each "get" call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

The `scha_control(3HA)` function also has a *tag* argument that indicates a control operation, but does not return information in an output argument.

The `scha_resource_setstatus(1HA)` command sets the `Status` and `Status_msg` properties of a resource that is managed by the RGM.

The man pages for the individual functions should be referred to for the macro values accepted as *tag* argument values for each function, and variable argument types for each *tag*. The types of output arguments are described in the next section.

There is one set of `scha_err_t` `enum`-type return values for the `scha` functions. The enum symbols, integer values, and meaning of the exit codes are described in RETURN VALUES.

The `scha_strerror(3HA)` function converts an `scha_err_t` code returned by an `scha` function to the appropriate error message.

Output Argument Data Types

`uint_t`
: An unsigned integer type. This type is defined in the system header file `<sys/types.h>`.

`boolean_t`
This type is defined in the system header file `<sys/types.h>`.

```
typedef enum { B_FALSE, B_TRUE } boolean_t;
```

`scha_switch_t`
An `enum` type that indicates an `On_Off_switch` or `Monitored_switch` resource property value.

```
typedef enum scha_switch {
    SCHA_SWITCH_DISABLED = 0,
    SCHA_SWITCH_ENABLED
} scha_switch_t;
```

`scha_rsstate_t`
An `enum` type that indicates a resource state.

```
typedef enum scha_rsstate {
    SCHA_RSSTATE_ONLINE = 0,
    SCHA_RSSTATE_OFFLINE,
    SCHA_RSSTATE_START_FAILED,
    SCHA_RSSTATE_STOP_FAILED,
    SCHA_RSSTATE_MONITOR_FAILED,
    SCHA_RSSTATE_ONLINE_NOT_MONITORED,
    SCHA_RSSTATE_STARTING,
    SCHA_RSSTATE_STOPPING
} scha_rsstate_t;
```

`scha_rgstate_t`
An `enum` type that indicates a resource group state.

```
typedef enum scha_rgstate {
    SCHA_RGSTATE_UNMANAGED = 0,
    SCHA_RGSTATE_ONLINE,
    SCHA_RGSTATE_OFFLINE,
    SCHA_RGSTATE_PENDING_ONLINE,
    SCHA_RGSTATE_PENDING_OFFLINE,
    SCHA_RGSTATE_ERROR_STOP_FAILED
    SCHA_RGSTATE_ONLINE_FAULTED,
    SCHA_RGSTATE_PENDING_ONLINE_BLOCKED
} scha_rgstate_t;
```

`scha_rgmode_t`
An `enum` type that indicates if the mode of a resource group is failover or scalable.

```
typedef enum scha_rgmode {
    RGMODE_NONE = 0,
    RGMODE_FAILOVER,
    RGMODE_SCALABLE
} scha_rgmode_t;
```

`scha_failover_mode_t`
An `enum` type that indicates a value for the `Failover_Mode` resource property.

```
typedef enum scha_failover_mode {
    SCHA_FOMODE_NONE = 0,
    SCHA_FOMODE_HARD,
    SCHA_FOMODE_SOFT,
```

```
    SCHA_FOMODE_RESTART_ONLY,
    SCHA_FOMODE_LOG_ONLY
} scha_failover_mode_t;
```

`scha_initnodes_flag_t`
An `enum` type that indicates a value for the `Init_nodes` resource type property.

```
typedef enum scha_initnodes_flag {
    SCHA_INFLAG_RG_PRIMARIES = 0,
    SCHA_INFLAG_RT_INSTALLED_NODES
} scha_initnodes_flag_t;
```

`scha_node_state_t`
An `enum` type that indicates whether a node is up or down.

```
typedef enum scha_node_state {
    SCHA_NODE_UP = 0,
    SCHA_NODE_DOWN
} scha_node_state_t;
```

`scha_str_array_t`
A structure that holds the value of a list of strings.

```
typedef struct scha_str_array {
    uint_t       array_cnt;
    boolean_t    is_ALL_value;
    char         **str_array;
} scha_str_array_t;
```

`array_cnt`	Gives the number elements in the list.
`is_ALL_value`	If a property is set to the "all" value, also known as the wild card or asterisk (*) character, `is_ALL_value` is set to `B_TRUE` and `str_array` is `NULL`. As a result, `str_array` is ignored.
`str_array`	A pointer to an array of `array_cnt` `strings`.

`scha_uint_array_t`
A structure that holds the value of a list of unsigned integers.

```
typedef struct scha_uint_array {
    uint_t    array_cnt;
    uint_t    *int_array;
} scha_uint_array_t;
```

`array_cnt`	The number of elements in the list.
`int_array`	A pointer to an array of `array_cnt` unsigned integers.

`scha_status_value_t`
The structure for returning the status and status message of a resource.

```
typedef struct scha_status_value {
    scha_rsstatus_t        status;
    char                   *status_msg;
} scha_status_value_t;
```

```
typedef enum scha_rsstatus {
    SCHA_RSSTATUS_ONLINE = 0,
    SCHA_RSSTATUS_OFFLINE,
    SCHA_RSSTATUS_FAULTED,
    SCHA_RSSTATUS_DEGRADED,
    SCHA_RSSTATUS_UNKNOWN
} scha_rsstatus_t;
```

`status` Holds an `enum` value that indicates the resource status as set by the resource monitor.

`scha_extprop_value_t`

The structure that is used for returning the value of an extension property.

The `prop_type` structure member indicates the type of the extension property and determines which element of the union is used for the `prop_type` field and the return values:

```
SCHA_PTYPE_STRING        val_str
SCHA_PTYPE_INT           val_int
SCHA_PTYPE_ENUM          val_enum
SCHA_PTYPE_BOOLEAN       val_boolean
SCHA_PTYPE_STRINGARRAY   val_strarray
```

```
typedef struct scha_extprop_value {
    scha_prop_type_t        prop_type;
    union {
          char              *val_str;
          int               val_int;
          char              *val_enum;
          boolean_t         val_boolean;
          scha_str_array_t  *val_strarray;
          } val;
} scha_extprop_value_t;
```

RETURN VALUES

The following is a list of the `scha_err_t` error numbers and the error codes returned by `scha_strerror`(3HA).

`0 SCHA_ERR_NOERR`	No error was found
`1 SCHA_ERR_NOMEM`	Not enough swap
`2 SCHA_ERR_HANDLE`	Invalid resource management handle
`3 SCHA_ERR_INVAL`	Invalid input argument
`4 SCHA_ERR_TAG`	Invalid API tag
`5 SCHA_ERR_RECONF`	Cluster is reconfiguring
`6 SCHA_ERR_ACCESS`	Permission denied
`7 SCHA_ERR_SEQID`	Resource, resource group, or resource type has been updated since last `scha_*_open` call
`8 SCHA_ERR_DEPEND`	Object dependency problem

9 SCHA_ERR_STATE	Object is in wrong state
10 SCHA_ERR_METHOD	Invalid method
11 SCHA_ERR_NODE	Invalid node
12 SCHA_ERR_RG	Invalid resource group
13 SCHA_ERR_RT	Invalid resource type
14 SCHA_ERR_RSRC	Invalid resource
15 SCHA_ERR_PROP	Invalid property
16 SCHA_ERR_CHECKS	Sanity checks failed
17 SCHA_ERR_RSTATUS	Bad resource status
18 SCHA_ERR_INTERNAL	Internal error was encountered
31 SCHA_ERR_TIMEOUT	Operation timed out
32 SCHA_ERR_FAIL	Failover attempt failed

FILES

/usr/cluster/include/scha.h	Include file
/usr/cluster/lib/libscha.so	Library

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

scha_cmds(1HA), scha_resource_setstatus(1HA)scha_cluster_get(3HA), scha_control(3HA), scha_resource_get(3HA), scha_resourcegroup_get(3HA), scha_resource_setstatus(3HA), scha_resourcetype_get(3HA), scha_strerror(3HA), attributes(5)

NAME

scha_cluster_open, scha_cluster_close, scha_cluster_get – cluster information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_cluster_open(scha_cluster_t *handle);

scha_err_t scha_cluster_get(scha_cluster_t handle, const char
    **tag, ...);

scha_err_t scha_cluster_close(scha_cluster_t handle);
```

DESCRIPTION

The `scha_cluster_open()`, `scha_cluster_get()`, and `scha_cluster_close()` functions are used together to obtain information about a cluster.

`scha_cluster_open()` initializes cluster access and returns an access handle to be used by `scha_cluster_get()`. The *handle* argument is the address of a variable to hold the value that is returned by the function call.

`scha_cluster_get()` accesses cluster information as indicated by the *tag* argument. The *handle* is a value that is returned from a prior call to `scha_cluster_open()`. The *tag* should be a string value defined by a macro in the `<scha_tags.h>` header file. The arguments that follow the tag depend on the value of *tag*.

An additional argument following the tag might be needed to indicate a cluster node from which the information is to be retrieved. The last argument in the argument list is to be of a type suitable to hold the information indicated by *tag*. This is the `out` argument for the cluster information. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_cluster_get()` remains intact until `scha_cluster_close()` is called on the handle that is used for `scha_cluster_get()`.

`scha_cluster_close()` takes a `handle` argument that is returned from a previous call to `scha_cluster_get()`. This function invalidates the handle and frees memory that is allocated to return values to `scha_cluster_get()` calls that were made with the handle. Note that memory, if needed to return a value, is allocated for each `get` call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros That You Can Use for *tag*

The macros that are defined in `<scha_tags.h>` that you can use as *tag* values follow. The type of the output argument and any additional arguments are indicated. Structure and `enum` types are described in `scha_calls(3HA)`.

`SCHA_NODENAME_LOCAL`
The output argument type is `char**`.

This macro returns the name of the cluster node where the function executed.

`SCHA_NODENAME_NODEID`
The output argument type is `char**`. An additional argument is of type `uint_t`. The additional argument is a numeric cluster node identifier.

This macro returns the name of the node indicated by the numeric identifier.

`SCHA_ALL_NODENAMES`
The output argument type is `scha_str_array_t**`.

This macro returns the names of all nodes in the cluster.

`SCHA_ALL_NODEIDS`
The output argument type is `scha_uint_array_t**`.

This macro returns numeric node identifiers for all the nodes in the cluster.

`SCHA_NODEID_LOCAL`
The output argument type is `uint_t*`.

This macro returns the numeric node identifier for the node where the command is executed.

`SCHA_NODEID_NODENAME`
The output argument type is `uint_t*`. An additional argument is of type `char *`. The macro requires an additional argument that is a name of a cluster node.

This macro returns the numeric node identifier of the node indicated by the name.

`SCHA_PRIVATELINK_HOSTNAME_LOCAL`
The output argument type is `char**`.

This macro returns the host name by which the node on which the command is run is addressed on the cluster interconnect.

`SCHA_PRIVATELINK_HOSTNAME_NODE`
The output argument type is `char**`. An additional argument is of type `char *`. This macro requires an additional unflagged argument that is the name of a cluster node.

This macro returns the host name by which the named node is addressed on the cluster interconnect.

`SCHA_ALL_PRIVATELINK_HOSTNAMES`
The output argument type is `scha_str_array_t**`.

This macro returns the host names for all cluster nodes by which the nodes are addressed on the cluster interconnect.

`SCHA_NODESTATE_LOCAL`
The output argument type is `scha_node_state_t*`.

This macro returns `SCHA_NODE_UP` or `SCHA_NODE_DOWN`, depending on the state of the node where the command is executed.

`SCHA_NODESTATE_NODE`
The output argument type is `scha_node_state_t*`. An additional argument is type `char*`. The macro requires an additional unflagged argument that is the name of a cluster node.

This macro returns SCHA_NODE_UP or SCHA_NODE_DOWN, depending on the state of the named node.

SCHA_SYSLOG_FACILITY
The output argument type is int*.

This macro returns the number of the syslog(3C) facility that the RGM uses for log messages. The value that is returned is 24, which corresponds to the LOG_DAEMON facility value.

SCHA_ALL_RESOURCEGROUPS
The output argument type is scha_str_array_t**.

This macro returns the names of all the resource groups that are being managed on the cluster.

SCHA_ALL_RESOURCETYPES
The output argument type is scha_str_array_t**.

This macro returns the names of all the resource types that are registered on the cluster.

SCHA_CLUSTERNAME
The output argument is type char**.

This macro returns the name of the cluster.

RETURN VALUES

The scha_cluster_open() function returns the following:

0	The function succeeded.
nonzero	The function failed.

ERRORS

SCHA_ERR_NOERR	Function succeeded.

See scha_calls(3HA) for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the scha_cluster_get(3HA) Function

The following example uses the scha_cluster_get() function to get the names of all cluster nodes and to find out whether the node is up or down.

```
#include <scha.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
        scha_err_t              err;
        scha_node_state_t       node_state;
        scha_str_array_t        *all_nodenames;
        scha_cluster_t          handle;
        int                     ix;
        const char              *str;

        err = scha_cluster_open(&handle);
```

EXAMPLE 1 Using the scha_cluster_get(3HA) Function *(Continued)*

```
        if (err != SCHA_ERR_NOERR) {
                fprintf(stderr, "FAILED: scha_cluster_open()0);
                exit(err);
        }

        err = scha_cluster_get(handle, SCHA_ALL_NODENAMES, &all_nodenames);
        if (err != SCHA_ERR_NOERR) {
                fprintf(stderr, "FAILED: scha_cluster_get()0);
                exit(err);
        }

        for (ix = 0; ix < all_nodenames->array_cnt; ix++) {
                err = scha_cluster_get(handle, SCHA_NODESTATE_NODE,
                    all_nodenames->str_array[ix], &node_state);
                if (err != SCHA_ERR_NOERR) {
                        fprintf(stderr, "FAILED: scha_cluster_get()"
                            "SCHA_NODESTATE_NODE0);
                        exit(err);
                }

                switch (node_state) {
                case SCHA_NODE_UP:
                        str = "UP";
                        break;
                case SCHA_NODE_DOWN:
                        str = "DOWN";
                        break;
                }

                printf("State of node: %s value: %s\
",
                    all_nodenames->str_array[ix], str);
        }
}
```

FILES

/usr/cluster/include/scha.h	Include file
/usr/cluster/lib/libscha.so	Library

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

scha_cluster_get(1HA), scha_calls(3HA), scha_cluster_getlogfacility(3HA), scha_cluster_getnodename(3HA), scha_strerror(3HA), syslog(3C), attributes(5)

NAME

scha_cluster_open, scha_cluster_close, scha_cluster_get – cluster information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_cluster_open(scha_cluster_t *handle);

scha_err_t scha_cluster_get(scha_cluster_t handle, const char
    **tag, ...);

scha_err_t scha_cluster_close(scha_cluster_t handle);
```

DESCRIPTION

The `scha_cluster_open()`, `scha_cluster_get()`, and `scha_cluster_close()` functions are used together to obtain information about a cluster.

`scha_cluster_open()` initializes cluster access and returns an access handle to be used by `scha_cluster_get()`. The *handle* argument is the address of a variable to hold the value that is returned by the function call.

`scha_cluster_get()` accesses cluster information as indicated by the *tag* argument. The *handle* is a value that is returned from a prior call to `scha_cluster_open()`. The *tag* should be a string value defined by a macro in the `<scha_tags.h>` header file. The arguments that follow the tag depend on the value of *tag*.

An additional argument following the tag might be needed to indicate a cluster node from which the information is to be retrieved. The last argument in the argument list is to be of a type suitable to hold the information indicated by *tag*. This is the `out` argument for the cluster information. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_cluster_get()` remains intact until `scha_cluster_close()` is called on the handle that is used for `scha_cluster_get()`.

`scha_cluster_close()` takes a `handle` argument that is returned from a previous call to `scha_cluster_get()`. This function invalidates the handle and frees memory that is allocated to return values to `scha_cluster_get()` calls that were made with the handle. Note that memory, if needed to return a value, is allocated for each `get` call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros That You Can Use for *tag*

The macros that are defined in `<scha_tags.h>` that you can use as *tag* values follow. The type of the output argument and any additional arguments are indicated. Structure and `enum` types are described in `scha_calls(3HA)`.

`SCHA_NODENAME_LOCAL`
The output argument type is `char**`.

This macro returns the name of the cluster node where the function executed.

`SCHA_NODENAME_NODEID`
The output argument type is `char**`. An additional argument is of type `uint_t`. The additional argument is a numeric cluster node identifier.

This macro returns the name of the node indicated by the numeric identifier.

`SCHA_ALL_NODENAMES`
The output argument type is `scha_str_array_t**`.

This macro returns the names of all nodes in the cluster.

`SCHA_ALL_NODEIDS`
The output argument type is `scha_uint_array_t**`.

This macro returns numeric node identifiers for all the nodes in the cluster.

`SCHA_NODEID_LOCAL`
The output argument type is `uint_t*`.

This macro returns the numeric node identifier for the node where the command is executed.

`SCHA_NODEID_NODENAME`
The output argument type is `uint_t*`. An additional argument is of type `char *`. The macro requires an additional argument that is a name of a cluster node.

This macro returns the numeric node identifier of the node indicated by the name.

`SCHA_PRIVATELINK_HOSTNAME_LOCAL`
The output argument type is `char**`.

This macro returns the host name by which the node on which the command is run is addressed on the cluster interconnect.

`SCHA_PRIVATELINK_HOSTNAME_NODE`
The output argument type is `char**`. An additional argument is of type `char *`. This macro requires an additional unflagged argument that is the name of a cluster node.

This macro returns the host name by which the named node is addressed on the cluster interconnect.

`SCHA_ALL_PRIVATELINK_HOSTNAMES`
The output argument type is `scha_str_array_t**`.

This macro returns the host names for all cluster nodes by which the nodes are addressed on the cluster interconnect.

`SCHA_NODESTATE_LOCAL`
The output argument type is `scha_node_state_t*`.

This macro returns `SCHA_NODE_UP` or `SCHA_NODE_DOWN`, depending on the state of the node where the command is executed.

`SCHA_NODESTATE_NODE`
The output argument type is `scha_node_state_t*`. An additional argument is type `char*`. The macro requires an additional unflagged argument that is the name of a cluster node.

This macro returns SCHA_NODE_UP or SCHA_NODE_DOWN, depending on the state of the named node.

SCHA_SYSLOG_FACILITY
The output argument type is int*.

This macro returns the number of the syslog(3C) facility that the RGM uses for log messages. The value that is returned is 24, which corresponds to the LOG_DAEMON facility value.

SCHA_ALL_RESOURCEGROUPS
The output argument type is scha_str_array_t**.

This macro returns the names of all the resource groups that are being managed on the cluster.

SCHA_ALL_RESOURCETYPES
The output argument type is scha_str_array_t**.

This macro returns the names of all the resource types that are registered on the cluster.

SCHA_CLUSTERNAME
The output argument is type char**.

This macro returns the name of the cluster.

RETURN VALUES

The scha_cluster_open() function returns the following:

0	The function succeeded.
nonzero	The function failed.

ERRORS

SCHA_ERR_NOERR	Function succeeded.

See scha_calls(3HA) for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the scha_cluster_get(3HA) Function

The following example uses the scha_cluster_get() function to get the names of all cluster nodes and to find out whether the node is up or down.

```
#include <scha.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
        scha_err_t              err;
        scha_node_state_t       node_state;
        scha_str_array_t        *all_nodenames;
        scha_cluster_t          handle;
        int                     ix;
        const char              *str;

        err = scha_cluster_open(&handle);
```

EXAMPLE 1 Using the scha_cluster_get(3HA) Function *(Continued)*

```
        if (err != SCHA_ERR_NOERR) {
                fprintf(stderr, "FAILED: scha_cluster_open()0);
                exit(err);
        }

        err = scha_cluster_get(handle, SCHA_ALL_NODENAMES, &all_nodenames);
        if (err != SCHA_ERR_NOERR) {
                fprintf(stderr, "FAILED: scha_cluster_get()0);
                exit(err);
        }

        for (ix = 0; ix < all_nodenames->array_cnt; ix++) {
                err = scha_cluster_get(handle, SCHA_NODESTATE_NODE,
                    all_nodenames->str_array[ix], &node_state);
                if (err != SCHA_ERR_NOERR) {
                        fprintf(stderr, "FAILED: scha_cluster_get()"
                            "SCHA_NODESTATE_NODE0);
                        exit(err);
                }

                switch (node_state) {
                case SCHA_NODE_UP:
                        str = "UP";
                        break;
                case SCHA_NODE_DOWN:
                        str = "DOWN";
                        break;
                }

                printf("State of node: %s value: %s\
",
                    all_nodenames->str_array[ix], str);
        }
}
```

FILES

/usr/cluster/include/scha.h	Include file
/usr/cluster/lib/libscha.so	Library

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

scha_cluster_get(1HA), scha_calls(3HA), scha_cluster_getlogfacility(3HA), scha_cluster_getnodename(3HA), scha_strerror(3HA), syslog(3C), attributes(5)

NAME

scha_cluster_getlogfacility – cluster log facility access

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_cluster_getlogfacility(int *logfacility);
```

DESCRIPTION

The `scha_cluster_getlogfacility()` function returns the system log facility number that is being used as the cluster log. The value is intended to be used with the Solaris `syslog`(3C) function by resource type implementations to record events and status messages to the cluster log.

The function returns an error status, and if successful, the facility number in the location pointed to by the *logfacility* argument.

RETURN VALUES

The `scha_cluster_getlogfacility()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls`(3HA) for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_cluster_getlogfacility()` Function

```
main()
{
    scha_err_t    err_code;
    int logfacility;

    err_code = scha_cluster_getlogfacility(&logfacility);

    if (err_code == SCHA_ERR_NOERR) {
        openlog("test resource", LOG_CONS, logfacility);
        syslog(LOG_INFO, "Access function call succeeded.");
    }
}
```

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`syslog`(3C), `scha_calls`(3HA), `scha_cluster_get`(3HA), `scha_strerror`(3HA), `attributes`(5)

NAME

scha_cluster_getnodename – local cluster node name access function

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_cluster_getnodename(char **nodename);
```

DESCRIPTION

The `scha_cluster_getnodename()` function returns the name of the cluster node on which the function is called. The cluster node name is not necessarily the same as the Solaris system name. The function returns an error status, and if successful, a string containing the node name in the location pointed to by the *nodename* argument. The *nodename* is set to `NULL` if the call fails. The caller of `scha_cluster_getnodename()` is responsible for freeing the memory allocated for the returned string value using the standard C library function `free`(3C). To avoid a core dump, only free the memory upon successful return of the function.

RETURN VALUES

The `scha_cluster_getnodename()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls`(3HA) for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_cluster_getnodename()` Function

```
scha_err_t  err_code;
char *nodename;
err_code = scha_cluster_getnodename(&nodename);
...
if( nodename != NULL ) free( nodename );
```

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`free`(3C), `scha_calls`(3HA), `scha_cluster_get`(3HA), `scha_strerror`(3HA), `attributes`(5)

NAME

scha_cluster_open, scha_cluster_close, scha_cluster_get – cluster information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_cluster_open(scha_cluster_t *handle);

scha_err_t scha_cluster_get(scha_cluster_t handle, const char
    **tag, ...);

scha_err_t scha_cluster_close(scha_cluster_t handle);
```

DESCRIPTION

The `scha_cluster_open()`, `scha_cluster_get()`, and `scha_cluster_close()` functions are used together to obtain information about a cluster.

`scha_cluster_open()` initializes cluster access and returns an access handle to be used by `scha_cluster_get()`. The *handle* argument is the address of a variable to hold the value that is returned by the function call.

`scha_cluster_get()` accesses cluster information as indicated by the *tag* argument. The *handle* is a value that is returned from a prior call to `scha_cluster_open()`. The *tag* should be a string value defined by a macro in the `<scha_tags.h>` header file. The arguments that follow the tag depend on the value of *tag*.

An additional argument following the tag might be needed to indicate a cluster node from which the information is to be retrieved. The last argument in the argument list is to be of a type suitable to hold the information indicated by *tag*. This is the `out` argument for the cluster information. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_cluster_get()` remains intact until `scha_cluster_close()` is called on the handle that is used for `scha_cluster_get()`.

`scha_cluster_close()` takes a `handle` argument that is returned from a previous call to `scha_cluster_get()`. This function invalidates the handle and frees memory that is allocated to return values to `scha_cluster_get()` calls that were made with the handle. Note that memory, if needed to return a value, is allocated for each `get` call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros That You Can Use for *tag*

The macros that are defined in `<scha_tags.h>` that you can use as *tag* values follow. The type of the output argument and any additional arguments are indicated. Structure and `enum` types are described in `scha_calls(3HA)`.

`SCHA_NODENAME_LOCAL`
The output argument type is `char**`.

This macro returns the name of the cluster node where the function executed.

`SCHA_NODENAME_NODEID`
The output argument type is `char**`. An additional argument is of type `uint_t`. The additional argument is a numeric cluster node identifier.

This macro returns the name of the node indicated by the numeric identifier.

`SCHA_ALL_NODENAMES`
The output argument type is `scha_str_array_t**`.

This macro returns the names of all nodes in the cluster.

`SCHA_ALL_NODEIDS`
The output argument type is `scha_uint_array_t**`.

This macro returns numeric node identifiers for all the nodes in the cluster.

`SCHA_NODEID_LOCAL`
The output argument type is `uint_t*`.

This macro returns the numeric node identifier for the node where the command is executed.

`SCHA_NODEID_NODENAME`
The output argument type is `uint_t*`. An additional argument is of type `char *`. The macro requires an additional argument that is a name of a cluster node.

This macro returns the numeric node identifier of the node indicated by the name.

`SCHA_PRIVATELINK_HOSTNAME_LOCAL`
The output argument type is `char**`.

This macro returns the host name by which the node on which the command is run is addressed on the cluster interconnect.

`SCHA_PRIVATELINK_HOSTNAME_NODE`
The output argument type is `char**`. An additional argument is of type `char *`. This macro requires an additional unflagged argument that is the name of a cluster node.

This macro returns the host name by which the named node is addressed on the cluster interconnect.

`SCHA_ALL_PRIVATELINK_HOSTNAMES`
The output argument type is `scha_str_array_t**`.

This macro returns the host names for all cluster nodes by which the nodes are addressed on the cluster interconnect.

`SCHA_NODESTATE_LOCAL`
The output argument type is `scha_node_state_t*`.

This macro returns `SCHA_NODE_UP` or `SCHA_NODE_DOWN`, depending on the state of the node where the command is executed.

`SCHA_NODESTATE_NODE`
The output argument type is `scha_node_state_t*`. An additional argument is type `char*`. The macro requires an additional unflagged argument that is the name of a cluster node.

This macro returns SCHA_NODE_UP or SCHA_NODE_DOWN, depending on the state of the named node.

SCHA_SYSLOG_FACILITY
The output argument type is int*.

This macro returns the number of the syslog(3C) facility that the RGM uses for log messages. The value that is returned is 24, which corresponds to the LOG_DAEMON facility value.

SCHA_ALL_RESOURCEGROUPS
The output argument type is scha_str_array_t**.

This macro returns the names of all the resource groups that are being managed on the cluster.

SCHA_ALL_RESOURCETYPES
The output argument type is scha_str_array_t**.

This macro returns the names of all the resource types that are registered on the cluster.

SCHA_CLUSTERNAME
The output argument is type char**.

This macro returns the name of the cluster.

RETURN VALUES

The scha_cluster_open() function returns the following:

0	The function succeeded.
nonzero	The function failed.

ERRORS

SCHA_ERR_NOERR	Function succeeded.

See scha_calls(3HA) for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the scha_cluster_get(3HA) Function

The following example uses the scha_cluster_get() function to get the names of all cluster nodes and to find out whether the node is up or down.

```
#include <scha.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
        scha_err_t              err;
        scha_node_state_t       node_state;
        scha_str_array_t        *all_nodenames;
        scha_cluster_t          handle;
        int                     ix;
        const char              *str;

        err = scha_cluster_open(&handle);
```

EXAMPLE 1 Using the scha_cluster_get(3HA) Function *(Continued)*

```
        if (err != SCHA_ERR_NOERR) {
                fprintf(stderr, "FAILED: scha_cluster_open()0);
                exit(err);
        }

        err = scha_cluster_get(handle, SCHA_ALL_NODENAMES, &all_nodenames);
        if (err != SCHA_ERR_NOERR) {
                fprintf(stderr, "FAILED: scha_cluster_get()0);
                exit(err);
        }

        for (ix = 0; ix < all_nodenames->array_cnt; ix++) {
                err = scha_cluster_get(handle, SCHA_NODESTATE_NODE,
                    all_nodenames->str_array[ix], &node_state);
                if (err != SCHA_ERR_NOERR) {
                        fprintf(stderr, "FAILED: scha_cluster_get()"
                            "SCHA_NODESTATE_NODE0);
                        exit(err);
                }

                switch (node_state) {
                case SCHA_NODE_UP:
                        str = "UP";
                        break;
                case SCHA_NODE_DOWN:
                        str = "DOWN";
                        break;
                }

                printf("State of node: %s value: %s\
",
                    all_nodenames->str_array[ix], str);
        }
}
```

FILES

/usr/cluster/include/scha.h	Include file
/usr/cluster/lib/libscha.so	Library

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

scha_cluster_get(1HA), scha_calls(3HA), scha_cluster_getlogfacility(3HA), scha_cluster_getnodename(3HA), scha_strerror(3HA), syslog(3C), attributes(5)

NAME

scha_control – resource group control request function

SYNOPSIS

```
cc [flags...]-I/usr/cluster/include file -L/usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_control(const char *tag, const char *rgname, const
     char *rname);
```

DESCRIPTION

The `scha_control()` function provides an interface to request the restart or relocation of a resource group or resource that is under the control of the Resource Group Manager (RGM) cluster facility. The command is intended to be used in resource monitors.

The setting of the `Failover_mode` property of the indicated resource might suppress the requested `scha_control` action. If `Failover_mode` is `RESTART_ONLY`, only `SCHA_RESOURCE_RESTART` is permitted. Other requests, including `SCHA_GIVEOVER`, `SCHA_CHECK_GIVEOVER`, `SCHA_RESTART`, and `SCHA_CHECK_RESTART`, return the `SCHA_ERR_CHECKS` exit code and the requested giveover or restart action is not executed, producing only a `syslog` message. If the `Retry_count` and `Retry_interval` properties are set on the resource, the number of resource restarts is limited to `Retry_count` attempts within the `Retry_interval`. If `Failover_mode` is `LOG_ONLY`, any `scha_control` request returns the `SCHA_ERR_CHECKS` exit code and the requested giveover or restart action is not executed, producing only a `syslog` message.

Macros That You Can Use for *tag*

The *tag* argument indicates whether the request is to restart or relocate the resource or group. This argument should be a string value that is defined by one of the following macros, which are defined in `<scha_tags.h>`:

`SCHA_CHECK_GIVEOVER`
: Perform all the same validity checks that would be done for a `SCHA_GIVEOVER` of the resource group named by the *rgname* argument, but do not actually relocate the resource group.

`SCHA_CHECK_RESTART`
: Perform all the same validity checks that would be done for an `SCHA_RESTART` of the resource group named by the *rgname* argument, but do not actually restart the resource group.

  The `SCHA_CHECK_GIVEOVER` and `SCHA_CHECK_RESTART` options are intended to be used by resource monitors that take direct action upon resources, for example, killing and restarting processes, rather than invoking `scha_control()` to perform a giveover or restart. If the check fails, the monitor should sleep and restart its probes rather than invoke its failover actions. See ERRORS.

  The *rgname* argument is the name of the resource group that is to be restarted or relocated. If the group is not online on the node where the request is made, the request is rejected.

  The *rname* argument is the name of a resource in the resource group. Presumably this is the resource whose monitor is making the `scha_control()` request. If the named resource is not in the resource group the request is rejected.

The exit code of the command indicates whether the requested action was rejected. If the request is accepted, the function does not return until the resource group or resource has completed going offline and back online. The fault monitor that called `scha_control()` might be stopped as a result of the resource group's going offline and so might never receive the return status of a successful request.

`SCHA_GIVEOVER`
Requests that the resource group named by the *rgname* argument be brought offline on the local node, and online again on a different node of the RGM's choosing. Note that, if the resource group is currently online on two or more nodes and there are no additional available nodes on which to bring the resource group online, it can be taken offline on the local node without being brought online elsewhere. The request might be rejected depending on the result of various checks. For example, a node might be rejected as a host because the group was brought offline due to a `SCHA_GIVEOVER` request on that node within the interval specified by the `Pingpong_interval` property.

If the cluster administrator configures the `RG_affinities` properties of one or more resource groups, and you issue a **`scha_control GIVEOVER`** request on one resource group, more than one resource group might be relocated as a result. The `RG_affinities` property is described in `rg_properties(5)`.

The `MONITOR_CHECK` method is called before the resource group that contains the resource is relocated to a new node as the result of a `scha_control(3HA)` or `scha_control(1HA)` request from a fault monitor.

The `MONITOR_CHECK` method may be called on any node that is a potential new master for the resource group. The `MONITOR_CHECK` method is intended to assess whether a node is running well enough to run a resource. The `MONITOR_CHECK` method must be implemented in such a way that it does not conflict with the running of another method concurrently.

`MONITOR_CHECK` failure vetoes the relocation of the resource group to the node where the callback was invoked.

`SCHA_IGNORE_FAILED_START`
Requests that failure of the currently executing `Prenet_start` or `Start` method should not cause a failover of the resource group, despite the setting of the `Failover_mode` property.

In other words, this value overrides the recovery action that is normally taken for a resource for which the `Failover_Mode` property is set to `SOFT` or `HARD` when that resource fails to start. Normally, the resource group fails over to a different node. Instead, the resource behaves as if `Failover_Mode` is set to `NONE`. The resource enters the `START_FAILED` state, and the resource group ends up in the `ONLINE_FAULTED` state, if no other errors occur.

This value is meaningful only when it is called from a `Start` or `Prenet_start` method that subsequently exits with a nonzero status or times out. This value is valid only for the current invocation of the `Start` or `Prenet_start` method.

`scha_control()` should be called with this value in a situation in which the `Start` method has determined that the resource cannot start successfully on another node. If this value is called by any other method, the error `SCHA_ERR_INVAL` is returned. This value prevents the "ping pong" failover of the resource group that would otherwise occur.

`SCHA_RESOURCE_IS_RESTARTED`
Request that the resource restart counter for the resource named by the `rname` argument be incremented on the local node, without actually restarting the resource.

A resource monitor that restarts a resource directly without calling `scha_control()` with the `RESOURCE_RESTART` option (for example, using `pmfadm`(1M)) can use this option to notify the RGM that the resource has been restarted. This fact is reflected in subsequent `scha_resource_get NUM_RESOURCE_RESTARTS` queries.

If the resource's type fails to declare the `Retry_interval` standard property, the `RESOURCE_IS_RESTARTED` option of `scha_control()` is not permitted and `scha_control()` returns error code 13 (`SCHA_ERR_RT`).

`SCHA_RESOURCE_RESTART`
Request that the resource named by the `rname` argument be brought offline and online again on the local node, without stopping any other resources in the resource group. The resource is stopped and restarted by applying the following sequence of methods to it on the local node:

```
MONITOR_STOP
STOP
START
MONITOR_START
```

If the resource's type does not declare a `MONITOR_STOP` and `MONITOR_START` method, only the `STOP` and `START` methods are invoked to perform the restart.The resource's type must declare a `START` and `STOP` method. If the resource's type does not declare both a `START` and `STOP` method, `scha_control()` fails with error code 13 (`SCHA_ERR_RT`).

If a method invocation fails while restarting the resource, the RGM might either set an error state, relocate the resource group, or reboot the node, depending on the setting of the `Failover_mode` property of the resource. For additional information, see the `Failover_mode` property in `r_properties`(5).

A resource monitor using this option to restart a resource can use the `NUM_RESOURCE_RESTARTS` query of `scha_resource_get()` to keep count of recent restart attempts.

The `RESOURCE_RESTART` function should be used with care by resource types that have `PRENET_START` or `POSTNET_STOP` methods. Only the `MONITOR_STOP`, `STOP`, `START`, and `MONITOR_START` methods are applied to the resource. Network address resources on which this resource implicitly depends is not restarted and remains online.

`SCHA_RESTART`
Request that the resource group named by the *rgname* argument be brought offline, then online again, without forcing relocation to a different node. The request may ultimately result in relocating the resource group if a resource in the group fails to restart. A resource monitor using this option to restart a resource group can use the `NUM_RG_RESTARTS` query of `scha_resource_get()` to keep count of recent restart attempts.

RETURN VALUES

The `scha_control()` function returns the following values:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	The function succeeded
`SCHA_ERR_CHECKS`	The request was rejected. The checks on relocation failed

See `scha_calls(3HA)` for a description of other error codes.

Normally, a fault monitor that receives an error code from `scha_control()` should sleep for awhile and then restart its probes, since some error conditions, for example, failover of a global device service causing disk resources to become temporarily unavailable, resolve themselves after awhile. Once the error condition has resolved, the resource itself might become healthy again, or if not, then a subsequent `scha_control()` request might succeed.

FILES

`</usr/cluster/include/scha.h>`	Include file
`/usr/cluster/lib/libscha.so`	Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO

`rt_callbacks(1HA)`, `scha_control(1HA)`, `scha_calls(3HA)`, `scha_resource_get(3HA)`, `scha_strerror(3HA)`, `attributes`(5), `r_properties(5)`, `rg_properties(5)`

NAME

scha_resource_open, scha_resource_close, scha_resource_get – resource information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resource_open(const char *rname, const char
    *rgname, scha_resource_t *handle);

scha_err_t scha_resource_close(scha_resource_t handle);

scha_err_t scha_resource_get(scha_resource_t handle, const char
    *tag, ...);
```

DESCRIPTION

The `scha_resource_open()`, `scha_resource_get()`, and `scha_resource_close()` functions are used together to access information about a resource that is managed by the Resource Group Manager (RGM) cluster facility.

`scha_resource_open()` initializes access of the resource and returns a handle to be used by `scha_resource_get()`.

The *rname* argument of `scha_resource_open()` names the resource to be accessed. The *rgname* argument is the name of the resource group in which the resource is configured. The *rgname* argument may be `NULL` if the group name is not known. However, the execution of the function is more efficient if it is provided. The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resource_get()` accesses resource information as indicated by the *tag* argument. The *tag* argument should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*. An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved, or other information that is specific to the tag. The last argument in the argument list is to be of a type that is suitable to hold the information that is indicated by *tag*. This argument is the `out` argument for the resource information. No value is returned for the `out` argument if the function fails.

Memory that is allocated to hold information returned by `scha_resource_get()` remains intact until `scha_resource_close()` is called on the handle used for the `scha_resource_get()`. Note that repeated calls to `scha_resource_get()` with the same handle and tag cause new memory to be allocated. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

`scha_resource_close()` takes a *handle* argument that is returned from a previous call to `scha_resource_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resource_get()` calls that were made with the handle.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resource_get()` follow.

The type of the output argument and any additional arguments are indicated. Structure and `enum` types are described in `scha_calls(3HA)`.

Tag Arguments

Macros that name resource properties are listed below. The value of the property of the resource is output. The SCHA_RESOURCE_STATE, SCHA_STATUS, SCHA_NUM_RG_RESTARTS, and SCHA_NUM_RESOURCE_RESTARTS properties refer to the value on the node where the command is executed (see r_properties(5)).

Extension properties
: These properties are declared in the RTR file of the resource's type. The implementation of the resource type defines these properties.

SCHA_AFFINITY_TIMEOUT
: The output argument type is int*.

SCHA_ALL_EXTENSIONS
: The output argument type is scha_str_array_t*.

SCHA_BOOT_TIMEOUT
: The output argument type is int*.

SCHA_CHEAP_PROBE_INTERVAL
: The output argument type is int*.

SCHA_EXTENSION
: The output argument type is scha_extprop_value_t*.

SCHA_FAILOVER_MODE
: The output argument type is scha_failover_mode_t*.

SCHA_FINI_TIMEOUT
: The output argument type is int*.

SCHA_GROUP
: The output argument type is char**.

SCHA_INIT_TIMEOUT
: The output argument type is int*.

SCHA_LOAD_BALANCING_POLICY
: The output argument type is char**.

SCHA_LOAD_BALANCING_WEIGHTS
: The output argument type is scha_str_array_t**.

SCHA_MONITOR_CHECK_TIMEOUT
: The output argument type is int*.

SCHA_MONITOR_START_TIMEOUT
: The output argument type is int*.

SCHA_MONITOR_STOP_TIMEOUT
: The output argument type is int*.

SCHA_MONITORED_SWITCH
: The output argument type is scha_switch_t*.

SCHA_NETWORK_RESOURCES_USED
: The output argument type is scha_str_array_t**.

SCHA_NUM_RESOURCE_RESTARTS
The output argument type is int*.

SCHA_NUM_RG_RESTARTS
The output argument type is int*.

SCHA_ON_OFF_SWITCH
The output argument type is scha_switch_t*.

SCHA_PORT_LIST
The output argument type is scha_str_array_t**.

SCHA_POSTNET_STOP_TIMEOUT
The output argument type is int*.

SCHA_PRENET_START_TIMEOUT
The output argument type is int*.

SCHA_R_DESCRIPTION
The output argument type is char**.

SCHA_RESOURCE_DEPENDENCIES
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_DEPENDENCIES_RESTART
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_DEPENDENCIES_WEAK
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_PROJECT_NAME
The output argument type is char**.

SCHA_RESOURCE_STATE
The output argument type is scha_rsstate_t*.

SCHA_RESOURCE_STATE_NODE
The output argument type is scha_rsstate_t*.

SCHA_RETRY_COUNT
The output argument type is int*.

SCHA_RETRY_INTERVAL
The output argument type is int*.

SCHA_SCALABLE
The output argument type is boolean_t*.

SCHA_START_TIMEOUT
The output argument type is int*.

SCHA_STATUS
The output argument type is scha_status_value_t**.

SCHA_STATUS_NODE
The output argument type is scha_status_value_t**.

`SCHA_STOP_TIMEOUT`
The output argument type is `int*`.

`SCHA_THOROUGH_PROBE_INTERVAL`
The output argument type is `int*`.

`SCHA_TYPE`
The output argument type is `char**`.

`SCHA_TYPE_VERSION`
The output argument type is `char**`.

`SCHA_UDP_AFFINITY`
The output argument type is `boolean_t*`.

`SCHA_UPDATE_TIMEOUT`
The output argument type is `int*`.

`SCHA_VALIDATE_TIMEOUT`
The output argument type is `int*`.

`SCHA_WEAK_AFFINITY`
The output argument type is `boolean_t*`.

Macros that name resource type properties are listed below. The value of the property of the resource's type is output. For descriptions of resource type properties, see `rt_properties(5)`.

`SCHA_API_VERSION`
The output argument type is `int*`.

`SCHA_BOOT`
The output argument type is `char**`.

`SCHA_FAILOVER`
The output argument type is `boolean_t*`.

`SCHA_FINI`
The output argument type is `char**`.

`SCHA_INIT`
The output argument type is `char**`.

`SCHA_INIT_NODES`
The output argument type is `scha_initnodes_flag_t*`.

`SCHA_INSTALLED_NODES`
The output argument type is `scha_str_array_t**`.

`SCHA_MONITOR_CHECK`
The output argument type is `char**`.

`SCHA_MONITOR_START`
The output argument type is `char**`.

`SCHA_MONITOR_STOP`
The output argument type is `char**`.

`SCHA_PKGLIST`
The output argument type is `scha_str_array_t**`.

`SCHA_POSTNET_STOP`
The output argument type is `char**`.

`SCHA_PRENET_START`
The output argument type is `char**`.

`SCHA_RT_BASEDIR`
The output argument type is `char**`.

`SCHA_RT_DESCRIPTION`
The output argument type is `char**`.

`SCHA_RT_SYSTEM`
The output argument type is `boolean_t*`.

`SCHA_RT_VERSION`
The output argument type is `char**`.

`SCHA_SINGLE_INSTANCE`
The output argument type is `boolean_t*`.

`SCHA_START`
The output argument type is `char**`.

`SCHA_STOP`
The output argument type is `char**`.

`SCHA_UPDATE`
The output argument type is `char**`.

`SCHA_VALIDATE`
The output argument type is `char**`.

RETURN VALUES

These functions return the following values:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls`(3HA) for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_resource_get()` Function

The following example uses `scha_resource_get()` to get the value of the `Retry_count` property of a resource, and the value of the extension property named `Loglevel`.

```
main() {
  #include <scha.h>

  scha_err_t err;
  int *retry_count_out;
```

EXAMPLE 1 Using the `scha_resource_get()` Function *(Continued)*

```
  scha_extprop_value_t *loglevel_out;
  scha_resource_t handle;

/* a configured resource */
  char * resource_name = "example_R";
/* resource group containing example_R */
  char * group_name = "example_RG";

  err = scha_resource_open(resource_name, group_name, &handle);

  err = scha_resource_get(handle, SCHA_RETRY_COUNT, &retry_count_out);

/* Given extension property must be defined in resourcetype RTR file. */
  err = scha_resource_get(handle, SCHA_EXTENSION, "LogLevel", &loglevel_out);

  err = scha_resource_close(handle);

  printf("The retry count for resource %s is %d\n", resource_name,
         retry_count_out);

  printf("The log level for resource %s is %d\n", resource_name,
         loglevel_out->val.val_int);
}
```

FILES

`</usr/cluster/include/scha.h>`	Include file
`/usr/cluster/lib/libscha.so`	Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO

`scha_resource_get`(1HA), `scha_calls`(3HA), `scha_strerror`(3HA), `attributes`(5), `r_properties`(5), `rt_properties`(5)

NAME

scha_resource_open, scha_resource_close, scha_resource_get – resource information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resource_open(const char *rname, const char
    *rgname, scha_resource_t *handle);

scha_err_t scha_resource_close(scha_resource_t handle);

scha_err_t scha_resource_get(scha_resource_t handle, const char
    *tag, ...);
```

DESCRIPTION

The `scha_resource_open()`, `scha_resource_get()`, and `scha_resource_close()` functions are used together to access information about a resource that is managed by the Resource Group Manager (RGM) cluster facility.

`scha_resource_open()` initializes access of the resource and returns a handle to be used by `scha_resource_get()`.

The *rname* argument of `scha_resource_open()` names the resource to be accessed. The *rgname* argument is the name of the resource group in which the resource is configured. The *rgname* argument may be `NULL` if the group name is not known. However, the execution of the function is more efficient if it is provided. The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resource_get()` accesses resource information as indicated by the *tag* argument. The *tag* argument should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*. An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved, or other information that is specific to the tag. The last argument in the argument list is to be of a type that is suitable to hold the information that is indicated by *tag*. This argument is the `out` argument for the resource information. No value is returned for the `out` argument if the function fails.

Memory that is allocated to hold information returned by `scha_resource_get()` remains intact until `scha_resource_close()` is called on the handle used for the `scha_resource_get()`. Note that repeated calls to `scha_resource_get()` with the same handle and tag cause new memory to be allocated. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

`scha_resource_close()` takes a *handle* argument that is returned from a previous call to `scha_resource_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resource_get()` calls that were made with the handle.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resource_get()` follow.

The type of the output argument and any additional arguments are indicated. Structure and `enum` types are described in `scha_calls(3HA)`.

Tag Arguments

Macros that name resource properties are listed below. The value of the property of the resource is output. The SCHA_RESOURCE_STATE, SCHA_STATUS, SCHA_NUM_RG_RESTARTS, and SCHA_NUM_RESOURCE_RESTARTS properties refer to the value on the node where the command is executed (see r_properties(5)).

Extension properties
These properties are declared in the RTR file of the resource's type. The implementation of the resource type defines these properties.

SCHA_AFFINITY_TIMEOUT
The output argument type is int*.

SCHA_ALL_EXTENSIONS
The output argument type is scha_str_array_t*.

SCHA_BOOT_TIMEOUT
The output argument type is int*.

SCHA_CHEAP_PROBE_INTERVAL
The output argument type is int*.

SCHA_EXTENSION
The output argument type is scha_extprop_value_t*.

SCHA_FAILOVER_MODE
The output argument type is scha_failover_mode_t*.

SCHA_FINI_TIMEOUT
The output argument type is int*.

SCHA_GROUP
The output argument type is char**.

SCHA_INIT_TIMEOUT
The output argument type is int*.

SCHA_LOAD_BALANCING_POLICY
The output argument type is char**.

SCHA_LOAD_BALANCING_WEIGHTS
The output argument type is scha_str_array_t**.

SCHA_MONITOR_CHECK_TIMEOUT
The output argument type is int*.

SCHA_MONITOR_START_TIMEOUT
The output argument type is int*.

SCHA_MONITOR_STOP_TIMEOUT
The output argument type is int*.

SCHA_MONITORED_SWITCH
The output argument type is scha_switch_t*.

SCHA_NETWORK_RESOURCES_USED
The output argument type is scha_str_array_t**.

SCHA_NUM_RESOURCE_RESTARTS
The output argument type is int*.

SCHA_NUM_RG_RESTARTS
The output argument type is int*.

SCHA_ON_OFF_SWITCH
The output argument type is scha_switch_t*.

SCHA_PORT_LIST
The output argument type is scha_str_array_t**.

SCHA_POSTNET_STOP_TIMEOUT
The output argument type is int*.

SCHA_PRENET_START_TIMEOUT
The output argument type is int*.

SCHA_R_DESCRIPTION
The output argument type is char**.

SCHA_RESOURCE_DEPENDENCIES
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_DEPENDENCIES_RESTART
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_DEPENDENCIES_WEAK
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_PROJECT_NAME
The output argument type is char**.

SCHA_RESOURCE_STATE
The output argument type is scha_rsstate_t*.

SCHA_RESOURCE_STATE_NODE
The output argument type is scha_rsstate_t*.

SCHA_RETRY_COUNT
The output argument type is int*.

SCHA_RETRY_INTERVAL
The output argument type is int*.

SCHA_SCALABLE
The output argument type is boolean_t*.

SCHA_START_TIMEOUT
The output argument type is int*.

SCHA_STATUS
The output argument type is scha_status_value_t**.

SCHA_STATUS_NODE
The output argument type is scha_status_value_t**.

`SCHA_STOP_TIMEOUT`
The output argument type is `int*`.

`SCHA_THOROUGH_PROBE_INTERVAL`
The output argument type is `int*`.

`SCHA_TYPE`
The output argument type is `char**`.

`SCHA_TYPE_VERSION`
The output argument type is `char**`.

`SCHA_UDP_AFFINITY`
The output argument type is `boolean_t*`.

`SCHA_UPDATE_TIMEOUT`
The output argument type is `int*`.

`SCHA_VALIDATE_TIMEOUT`
The output argument type is `int*`.

`SCHA_WEAK_AFFINITY`
The output argument type is `boolean_t*`.

Macros that name resource type properties are listed below. The value of the property of the resource's type is output. For descriptions of resource type properties, see `rt_properties(5)`.

`SCHA_API_VERSION`
The output argument type is `int*`.

`SCHA_BOOT`
The output argument type is `char**`.

`SCHA_FAILOVER`
The output argument type is `boolean_t*`.

`SCHA_FINI`
The output argument type is `char**`.

`SCHA_INIT`
The output argument type is `char**`.

`SCHA_INIT_NODES`
The output argument type is `scha_initnodes_flag_t*`.

`SCHA_INSTALLED_NODES`
The output argument type is `scha_str_array_t**`.

`SCHA_MONITOR_CHECK`
The output argument type is `char**`.

`SCHA_MONITOR_START`
The output argument type is `char**`.

`SCHA_MONITOR_STOP`
The output argument type is `char**`.

`SCHA_PKGLIST`
The output argument type is `scha_str_array_t**`.

`SCHA_POSTNET_STOP`
The output argument type is `char**`.

`SCHA_PRENET_START`
The output argument type is `char**`.

`SCHA_RT_BASEDIR`
The output argument type is `char**`.

`SCHA_RT_DESCRIPTION`
The output argument type is `char**`.

`SCHA_RT_SYSTEM`
The output argument type is `boolean_t*`.

`SCHA_RT_VERSION`
The output argument type is `char**`.

`SCHA_SINGLE_INSTANCE`
The output argument type is `boolean_t*`.

`SCHA_START`
The output argument type is `char**`.

`SCHA_STOP`
The output argument type is `char**`.

`SCHA_UPDATE`
The output argument type is `char**`.

`SCHA_VALIDATE`
The output argument type is `char**`.

RETURN VALUES

These functions return the following values:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_resource_get()` Function

The following example uses `scha_resource_get()` to get the value of the `Retry_count` property of a resource, and the value of the extension property named `Loglevel`.

```
main() {
  #include <scha.h>

  scha_err_t err;
  int *retry_count_out;
```

EXAMPLE 1 Using the `scha_resource_get()` Function *(Continued)*

```
  scha_extprop_value_t *loglevel_out;
  scha_resource_t handle;

/* a configured resource */
  char * resource_name = "example_R";
/* resource group containing example_R */
  char * group_name = "example_RG";

  err = scha_resource_open(resource_name, group_name, &handle);

  err = scha_resource_get(handle, SCHA_RETRY_COUNT, &retry_count_out);

/* Given extension property must be defined in resourcetype RTR file. */
  err = scha_resource_get(handle, SCHA_EXTENSION, "LogLevel", &loglevel_out);

  err = scha_resource_close(handle);

  printf("The retry count for resource %s is %d\n", resource_name,
         retry_count_out);

  printf("The log level for resource %s is %d\n", resource_name,
         loglevel_out->val.val_int);
}
```

FILES

`</usr/cluster/include/scha.h>`	Include file
`/usr/cluster/lib/libscha.so`	Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO

`scha_resource_get`(1HA), `scha_calls`(3HA), `scha_strerror`(3HA), `attributes`(5), `r_properties`(5), `rt_properties`(5)

NAME

scha_resourcegroup_open, scha_resourcegroup_close, scha_resourcegroup_get – resource information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resourcegroup_open(const char *rgname,
     scha_resourcegroup_t *handle);

scha_err_t scha_resourcegroup_close(scha_resourcegroup_t *handle);

scha_err_t scha_resourcegroup_get(scha_resourcegroup_t *handle,
     const char *tag...);
```

DESCRIPTION

The `scha_resourcegroup_open()`, `scha_resourcegroup_get()`, and `scha_resourcegroup_close()` functions are used together to access information about a resource group that is managed by the Resource Group Manager (RGM) cluster facility.

`scha_resourcegroup_open()` initializes access of the resource group and returns a handle to be used by `scha_resourcegroup_get()`.

The *rgname* argument names the resource group to be accessed.

The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resourcegroup_get()` accesses resource group information as indicated by the *tag* argument. The *tag* should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*. An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved.

The last argument in the argument list is to be of a type suitable to hold the information indicated by *tag*. This is the `out` argument for the resource group information that is to be retrieved. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_resourcegroup_get()` remains intact until `scha_resourcegroup_close()` is called on the handle used for `scha_resourcegroup_get()`.

`scha_resourcegroup_close()` takes a *handle* argument returned from a previous call to `scha_resourcegroup_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resourcegroup_get()` calls that were made with the handle. Note that memory, if needed to return a value, is allocated for each `get` call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resourcegroup_get()` follow. The type of the output argument and any additional arguments are indicated. Structure and enum types are described in `scha_calls`(3HA).

Tag Arguments

Macros naming resource group properties are listed below. The value of the property of the resource group is output. The `RG_STATE` property refers to the value on the node where the function is called.

`SCHA_DESIRED_PRIMARIES`
The output argument type is `int*`.

`SCHA_FAILBACK`
The output argument type is `boolean_t*`.

`SCHA_GLOBAL_RESOURCES_USED`
The output argument type is `scha_str_array_t**`.

`SCHA_IMPL_NET_DEPEND`
The output argument type is `boolean_t*`.

`SCHA_MAXIMUM_PRIMARIES`
The output argument type is `int*`.

`SCHA_NODELIST`
The output argument type is `scha_str_array_t**`.

`SCHA_PATHPREFIX`
The output argument type is `char**`.

`SCHA_PINGPONG_INTERVAL`
The output argument type is `int*`.

`SCHA_RESOURCE_LIST`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_AUTO_START`
The output argument type is `boolean_t*`.

`SCHA_RG_DEPENDENCIES`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_DESCRIPTION`
The output argument type is `char**`.

`SCHA_RG_IS_FROZEN`
The output argument type is `boolean_t*`.

`SCHA_RG_MODE`
The output argument type is `scha_rgmode_t*`.

`SCHA_RG_PROJECT_NAME`
The output argument type is `char**`.

`SCHA_RG_STATE`
The output argument type is `scha_rgstate_t*`.

`SCHA_RG_STATE_NODE`
The output argument type is `scha_rgstate_t*`. An additional argument type is `char*`. The additional argument names a cluster node and returns the state of the resource group on that node.

`SCHA_RG_SYSTEM`
The output argument type is `boolean_t*`.

RETURN VALUES

These functions return the following:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_resourcegroup_get()` Function

The following example uses `scha_resourcegroup_get()` to get the list of resources in the resource group `example_RG`.

```
main() {
   #include <scha.h>

   scha_err_t err;
   scha_str_array_t *resource_list;
   scha_resourcegroup_t handle;
   int ix;

   char * rgname = "example_RG";

   err = scha_resourcegroup_open(rgname, &handle);

   err = scha_resourcegroup_get(handle, SCHA_RESOURCE_LIST, &resource_list);

   if (err == SCHA_ERR_NOERR) {
    for (ix = 0; ix < resource_list->array_cnt; ix++) {
        printf("Group: %s contains resource %s\
", rgname,
                resource_list->str_array[ix]);
        }
    }

 err = scha_resourcegroup_close(handle);     /* resource_list memory freed */
}
```

FILES

`/usr/cluster/include/scha.h`	Include file
`/usr/cluster/lib/libscha.so`	Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO | scha_resourcegroup_get(1HA), scha_calls(3HA), attributes(5)

NAME

scha_resourcegroup_open, scha_resourcegroup_close, scha_resourcegroup_get – resource information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resourcegroup_open(const char *rgname,
    scha_resourcegroup_t *handle);

scha_err_t scha_resourcegroup_close(scha_resourcegroup_t *handle);

scha_err_t scha_resourcegroup_get(scha_resourcegroup_t *handle,
    const char *tag...);
```

DESCRIPTION

The `scha_resourcegroup_open()`, `scha_resourcegroup_get()`, and `scha_resourcegroup_close()` functions are used together to access information about a resource group that is managed by the Resource Group Manager (RGM) cluster facility.

`scha_resourcegroup_open()` initializes access of the resource group and returns a handle to be used by `scha_resourcegroup_get()`.

The *rgname* argument names the resource group to be accessed.

The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resourcegroup_get()` accesses resource group information as indicated by the *tag* argument. The *tag* should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*. An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved.

The last argument in the argument list is to be of a type suitable to hold the information indicated by *tag*. This is the `out` argument for the resource group information that is to be retrieved. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_resourcegroup_get()` remains intact until `scha_resourcegroup_close()` is called on the handle used for `scha_resourcegroup_get()`.

`scha_resourcegroup_close()` takes a *handle* argument returned from a previous call to `scha_resourcegroup_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resourcegroup_get()` calls that were made with the handle. Note that memory, if needed to return a value, is allocated for each `get` call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resourcegroup_get()` follow. The type of the output argument and any additional arguments are indicated. Structure and enum types are described in `scha_calls`(3HA).

Tag Arguments

Macros naming resource group properties are listed below. The value of the property of the resource group is output. The `RG_STATE` property refers to the value on the node where the function is called.

`SCHA_DESIRED_PRIMARIES`
The output argument type is `int*`.

`SCHA_FAILBACK`
The output argument type is `boolean_t*`.

`SCHA_GLOBAL_RESOURCES_USED`
The output argument type is `scha_str_array_t**`.

`SCHA_IMPL_NET_DEPEND`
The output argument type is `boolean_t*`.

`SCHA_MAXIMUM_PRIMARIES`
The output argument type is `int*`.

`SCHA_NODELIST`
The output argument type is `scha_str_array_t**`.

`SCHA_PATHPREFIX`
The output argument type is `char**`.

`SCHA_PINGPONG_INTERVAL`
The output argument type is `int*`.

`SCHA_RESOURCE_LIST`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_AUTO_START`
The output argument type is `boolean_t*`.

`SCHA_RG_DEPENDENCIES`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_DESCRIPTION`
The output argument type is `char**`.

`SCHA_RG_IS_FROZEN`
The output argument type is `boolean_t*`.

`SCHA_RG_MODE`
The output argument type is `scha_rgmode_t*`.

`SCHA_RG_PROJECT_NAME`
The output argument type is `char**`.

`SCHA_RG_STATE`
The output argument type is `scha_rgstate_t*`.

`SCHA_RG_STATE_NODE`
The output argument type is `scha_rgstate_t*`. An additional argument type is `char*`. The additional argument names a cluster node and returns the state of the resource group on that node.

`SCHA_RG_SYSTEM`
The output argument type is `boolean_t*`.

RETURN VALUES

These functions return the following:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_resourcegroup_get()` Function

The following example uses `scha_resourcegroup_get()` to get the list of resources in the resource group `example_RG`.

```
main() {
   #include <scha.h>

   scha_err_t err;
   scha_str_array_t *resource_list;
   scha_resourcegroup_t handle;
   int ix;

   char * rgname = "example_RG";

   err = scha_resourcegroup_open(rgname, &handle);

   err = scha_resourcegroup_get(handle, SCHA_RESOURCE_LIST, &resource_list);

   if (err == SCHA_ERR_NOERR) {
    for (ix = 0; ix < resource_list->array_cnt; ix++) {
        printf("Group: %s contains resource %s\
", rgname,
                resource_list->str_array[ix]);
        }
    }

 err = scha_resourcegroup_close(handle);     /* resource_list memory freed */
}
```

FILES

`/usr/cluster/include/scha.h`	Include file
`/usr/cluster/lib/libscha.so`	Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO | scha_resourcegroup_get(1HA), scha_calls(3HA), attributes(5)

NAME

scha_resourcegroup_open, scha_resourcegroup_close, scha_resourcegroup_get – resource information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resourcegroup_open(const char *rgname,
    scha_resourcegroup_t *handle);

scha_err_t scha_resourcegroup_close(scha_resourcegroup_t *handle);

scha_err_t scha_resourcegroup_get(scha_resourcegroup_t *handle,
    const char *tag...);
```

DESCRIPTION

The `scha_resourcegroup_open()`, `scha_resourcegroup_get()`, and `scha_resourcegroup_close()` functions are used together to access information about a resource group that is managed by the Resource Group Manager (RGM) cluster facility.

`scha_resourcegroup_open()` initializes access of the resource group and returns a handle to be used by `scha_resourcegroup_get()`.

The *rgname* argument names the resource group to be accessed.

The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resourcegroup_get()` accesses resource group information as indicated by the *tag* argument. The *tag* should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*. An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved.

The last argument in the argument list is to be of a type suitable to hold the information indicated by *tag*. This is the `out` argument for the resource group information that is to be retrieved. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_resourcegroup_get()` remains intact until `scha_resourcegroup_close()` is called on the handle used for `scha_resourcegroup_get()`.

`scha_resourcegroup_close()` takes a *handle* argument returned from a previous call to `scha_resourcegroup_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resourcegroup_get()` calls that were made with the handle. Note that memory, if needed to return a value, is allocated for each `get` call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resourcegroup_get()` follow. The type of the output argument and any additional arguments are indicated. Structure and enum types are described in `scha_calls`(3HA).

Tag Arguments

Macros naming resource group properties are listed below. The value of the property of the resource group is output. The `RG_STATE` property refers to the value on the node where the function is called.

`SCHA_DESIRED_PRIMARIES`
The output argument type is `int*`.

`SCHA_FAILBACK`
The output argument type is `boolean_t*`.

`SCHA_GLOBAL_RESOURCES_USED`
The output argument type is `scha_str_array_t**`.

`SCHA_IMPL_NET_DEPEND`
The output argument type is `boolean_t*`.

`SCHA_MAXIMUM_PRIMARIES`
The output argument type is `int*`.

`SCHA_NODELIST`
The output argument type is `scha_str_array_t**`.

`SCHA_PATHPREFIX`
The output argument type is `char**`.

`SCHA_PINGPONG_INTERVAL`
The output argument type is `int*`.

`SCHA_RESOURCE_LIST`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_AUTO_START`
The output argument type is `boolean_t*`.

`SCHA_RG_DEPENDENCIES`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_DESCRIPTION`
The output argument type is `char**`.

`SCHA_RG_IS_FROZEN`
The output argument type is `boolean_t*`.

`SCHA_RG_MODE`
The output argument type is `scha_rgmode_t*`.

`SCHA_RG_PROJECT_NAME`
The output argument type is `char**`.

`SCHA_RG_STATE`
The output argument type is `scha_rgstate_t*`.

`SCHA_RG_STATE_NODE`
The output argument type is `scha_rgstate_t*`. An additional argument type is `char*`. The additional argument names a cluster node and returns the state of the resource group on that node.

`SCHA_RG_SYSTEM`
The output argument type is `boolean_t*`.

RETURN VALUES

These functions return the following:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_resourcegroup_get()` Function

The following example uses `scha_resourcegroup_get()` to get the list of resources in the resource group `example_RG`.

```
main() {
   #include <scha.h>

   scha_err_t err;
   scha_str_array_t *resource_list;
   scha_resourcegroup_t handle;
   int ix;

   char * rgname = "example_RG";

   err = scha_resourcegroup_open(rgname, &handle);

   err = scha_resourcegroup_get(handle, SCHA_RESOURCE_LIST, &resource_list);

   if (err == SCHA_ERR_NOERR) {
    for (ix = 0; ix < resource_list->array_cnt; ix++) {
        printf("Group: %s contains resource %s\
", rgname,
                resource_list->str_array[ix]);
        }
    }

 err = scha_resourcegroup_close(handle);     /* resource_list memory freed */
}
```

FILES

`/usr/cluster/include/scha.h`	Include file
`/usr/cluster/lib/libscha.so`	Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO | scha_resourcegroup_get(1HA), scha_calls(3HA), attributes(5)

NAME

scha_resource_open, scha_resource_close, scha_resource_get – resource information access functions

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resource_open(const char *rname, const char
    *rgname, scha_resource_t *handle);

scha_err_t scha_resource_close(scha_resource_t handle);

scha_err_t scha_resource_get(scha_resource_t handle, const char
    *tag, ...);
```

DESCRIPTION

The `scha_resource_open()`, `scha_resource_get()`, and `scha_resource_close()` functions are used together to access information about a resource that is managed by the Resource Group Manager (RGM) cluster facility.

`scha_resource_open()` initializes access of the resource and returns a handle to be used by `scha_resource_get()`.

The *rname* argument of `scha_resource_open()` names the resource to be accessed. The *rgname* argument is the name of the resource group in which the resource is configured. The *rgname* argument may be `NULL` if the group name is not known. However, the execution of the function is more efficient if it is provided. The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resource_get()` accesses resource information as indicated by the *tag* argument. The *tag* argument should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*. An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved, or other information that is specific to the tag. The last argument in the argument list is to be of a type that is suitable to hold the information that is indicated by *tag*. This argument is the `out` argument for the resource information. No value is returned for the `out` argument if the function fails.

Memory that is allocated to hold information returned by `scha_resource_get()` remains intact until `scha_resource_close()` is called on the handle used for the `scha_resource_get()`. Note that repeated calls to `scha_resource_get()` with the same handle and tag cause new memory to be allocated. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

`scha_resource_close()` takes a *handle* argument that is returned from a previous call to `scha_resource_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resource_get()` calls that were made with the handle.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resource_get()` follow.

The type of the output argument and any additional arguments are indicated. Structure and `enum` types are described in `scha_calls(3HA)`.

Tag Arguments

Macros that name resource properties are listed below. The value of the property of the resource is output. The SCHA_RESOURCE_STATE, SCHA_STATUS, SCHA_NUM_RG_RESTARTS, and SCHA_NUM_RESOURCE_RESTARTS properties refer to the value on the node where the command is executed (see r_properties(5)).

Extension properties
These properties are declared in the RTR file of the resource's type. The implementation of the resource type defines these properties.

SCHA_AFFINITY_TIMEOUT
The output argument type is int*.

SCHA_ALL_EXTENSIONS
The output argument type is scha_str_array_t*.

SCHA_BOOT_TIMEOUT
The output argument type is int*.

SCHA_CHEAP_PROBE_INTERVAL
The output argument type is int*.

SCHA_EXTENSION
The output argument type is scha_extprop_value_t*.

SCHA_FAILOVER_MODE
The output argument type is scha_failover_mode_t*.

SCHA_FINI_TIMEOUT
The output argument type is int*.

SCHA_GROUP
The output argument type is char**.

SCHA_INIT_TIMEOUT
The output argument type is int*.

SCHA_LOAD_BALANCING_POLICY
The output argument type is char**.

SCHA_LOAD_BALANCING_WEIGHTS
The output argument type is scha_str_array_t**.

SCHA_MONITOR_CHECK_TIMEOUT
The output argument type is int*.

SCHA_MONITOR_START_TIMEOUT
The output argument type is int*.

SCHA_MONITOR_STOP_TIMEOUT
The output argument type is int*.

SCHA_MONITORED_SWITCH
The output argument type is scha_switch_t*.

SCHA_NETWORK_RESOURCES_USED
The output argument type is scha_str_array_t**.

SCHA_NUM_RESOURCE_RESTARTS
The output argument type is int*.

SCHA_NUM_RG_RESTARTS
The output argument type is int*.

SCHA_ON_OFF_SWITCH
The output argument type is scha_switch_t*.

SCHA_PORT_LIST
The output argument type is scha_str_array_t**.

SCHA_POSTNET_STOP_TIMEOUT
The output argument type is int*.

SCHA_PRENET_START_TIMEOUT
The output argument type is int*.

SCHA_R_DESCRIPTION
The output argument type is char**.

SCHA_RESOURCE_DEPENDENCIES
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_DEPENDENCIES_RESTART
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_DEPENDENCIES_WEAK
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_PROJECT_NAME
The output argument type is char**.

SCHA_RESOURCE_STATE
The output argument type is scha_rsstate_t*.

SCHA_RESOURCE_STATE_NODE
The output argument type is scha_rsstate_t*.

SCHA_RETRY_COUNT
The output argument type is int*.

SCHA_RETRY_INTERVAL
The output argument type is int*.

SCHA_SCALABLE
The output argument type is boolean_t*.

SCHA_START_TIMEOUT
The output argument type is int*.

SCHA_STATUS
The output argument type is scha_status_value_t**.

SCHA_STATUS_NODE
The output argument type is scha_status_value_t**.

`SCHA_STOP_TIMEOUT`
The output argument type is `int*`.

`SCHA_THOROUGH_PROBE_INTERVAL`
The output argument type is `int*`.

`SCHA_TYPE`
The output argument type is `char**`.

`SCHA_TYPE_VERSION`
The output argument type is `char**`.

`SCHA_UDP_AFFINITY`
The output argument type is `boolean_t*`.

`SCHA_UPDATE_TIMEOUT`
The output argument type is `int*`.

`SCHA_VALIDATE_TIMEOUT`
The output argument type is `int*`.

`SCHA_WEAK_AFFINITY`
The output argument type is `boolean_t*`.

Macros that name resource type properties are listed below. The value of the property of the resource's type is output. For descriptions of resource type properties, see `rt_properties(5)`.

`SCHA_API_VERSION`
The output argument type is `int*`.

`SCHA_BOOT`
The output argument type is `char**`.

`SCHA_FAILOVER`
The output argument type is `boolean_t*`.

`SCHA_FINI`
The output argument type is `char**`.

`SCHA_INIT`
The output argument type is `char**`.

`SCHA_INIT_NODES`
The output argument type is `scha_initnodes_flag_t*`.

`SCHA_INSTALLED_NODES`
The output argument type is `scha_str_array_t**`.

`SCHA_MONITOR_CHECK`
The output argument type is `char**`.

`SCHA_MONITOR_START`
The output argument type is `char**`.

`SCHA_MONITOR_STOP`
The output argument type is `char**`.

`SCHA_PKGLIST`
The output argument type is `scha_str_array_t**`.

`SCHA_POSTNET_STOP`
The output argument type is `char**`.

`SCHA_PRENET_START`
The output argument type is `char**`.

`SCHA_RT_BASEDIR`
The output argument type is `char**`.

`SCHA_RT_DESCRIPTION`
The output argument type is `char**`.

`SCHA_RT_SYSTEM`
The output argument type is `boolean_t*`.

`SCHA_RT_VERSION`
The output argument type is `char**`.

`SCHA_SINGLE_INSTANCE`
The output argument type is `boolean_t*`.

`SCHA_START`
The output argument type is `char**`.

`SCHA_STOP`
The output argument type is `char**`.

`SCHA_UPDATE`
The output argument type is `char**`.

`SCHA_VALIDATE`
The output argument type is `char**`.

RETURN VALUES

These functions return the following values:

`0`	The function succeeded.
nonzero	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls`(3HA) for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_resource_get()` Function

The following example uses `scha_resource_get()` to get the value of the `Retry_count` property of a resource, and the value of the extension property named `Loglevel`.

```
main() {
  #include <scha.h>

  scha_err_t err;
  int *retry_count_out;
```

EXAMPLE 1 Using the `scha_resource_get()` Function *(Continued)*

```
  scha_extprop_value_t *loglevel_out;
  scha_resource_t handle;

/* a configured resource */
  char * resource_name = "example_R";
/* resource group containing example_R */
  char * group_name = "example_RG";

  err = scha_resource_open(resource_name, group_name, &handle);

  err = scha_resource_get(handle, SCHA_RETRY_COUNT, &retry_count_out);

/* Given extension property must be defined in resourcetype RTR file. */
  err = scha_resource_get(handle, SCHA_EXTENSION, "LogLevel", &loglevel_out);

  err = scha_resource_close(handle);

  printf("The retry count for resource %s is %d\n", resource_name,
         retry_count_out);

  printf("The log level for resource %s is %d\n", resource_name,
         loglevel_out->val.val_int);
}
```

FILES

`</usr/cluster/include/scha.h>`	Include file
`/usr/cluster/lib/libscha.so`	Library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscdev`
Interface Stability	Evolving

SEE ALSO

`scha_resource_get`(1HA), `scha_calls`(3HA), `scha_strerror`(3HA), `attributes`(5), `r_properties`(5), `rt_properties`(5)

NAME

scha_resource_setstatus – function to set resource status

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resource_setstatus(const char *rname, const char
    *rgname, scha_rsstatus_t status, const char *status_msg);
```

DESCRIPTION

The `scha_resource_setstatus()` function sets the `Status` and `Status_msg` property of a resource that is managed by the Resource Group Manager (RGM) cluster facility. It is intended to be used by the resource's monitor to indicate the resource's state as perceived by the monitor.

The *rname* argument names the resource whose status is to be set.

The *rgname* argument is the name of the group containing the resource.

The *status* is an enum value of type `scha_rsstatus_t`: `SCHA_RSSTATUS_OK`, `SCHA_RSSTATUS_OFFLINE`, `SCHA_RSSTATUS_FAULTED`, `SCHA_RSSTATUS_DEGRADED` or `SCHA_RSSTATUS_UNKNOWN`.

The *status_msg* argument is the new value for the `Status_msg` property and may be `NULL`.

A successful call to `scha_resource_setstatus()` causes the `Status` and `Status_msg` properties of the resource to be updated to the supplied values. The update of the resource status is logged in the cluster system log and is visible to cluster administration tools.

RETURN VALUES

The `scha_resosurce_setstatus()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

EXAMPLES

EXAMPLE 1 Using the `scha_resource_setstatus()` Function

```
#include <scha.h>

scha_err_t err_code;
const char *rname = "example_R";
const char *rgname = "example_RG";

err_code = scha_resource_setstatus(rname, rgname,
           SCHA_RSSTATUS_OK, "No problems");
```

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO `scha_resource_setstatus`(1HA), `scha_calls`(3HA), `scha_strerror`(3HA), `attributes`(5)

NAME

scha_resourcetype_open, scha_resourcetype_close, scha_resourcetype_get – resource type information access functions.

SYNOPSIS

```
cc [flags...] -I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resourcetype_open(const char *rtname,
    scha_resourcetype_t *handle);

scha_err_t scha_resourcetype_close(scha_resourcetype_t handle);

scha_err_t scha_resourcetype_get(scha_resourcetype_t handle, const
    char *tag...);
```

DESCRIPTION

The `scha_resourcetype_open()`, `scha_resourcetype_get()`, and `scha_resourcetype_close()` functions are used together to access information on a resource type that is used by the Resource Group Manager (RGM) cluster facility.

`scha_resourcetype_open()` initializes access of the resource type and returns a handle to be used by `scha_resourcetype_get()`.

The *rtname* argument of `scha_resourcetype_open()` names the resource type to be accessed.

The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resourcetype_get()` accesses resource type information as indicated by the *tag* argument. The *tag* argument should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*.

An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved, or other information specific to the tag. The last argument in the argument list is to be of a type suitable type to hold the information indicated by *tag*. This is the "out" argument for the resource type information. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_resourcetype_get()` remains intact until `scha_resourcetype_close()` is called on the handle used for `scha_resourcetype_get()`.

`scha_resourcetype_close()` takes a *handle* argument returned from a previous call to `scha_resourcetype_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resourcetype_get()` calls that were made with the handle. Note that, memory, if needed to return a value, is allocated for each "get" call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resourcetype_get()` follow. The type of the output argument and any additional arguments are indicated. Structure and enum types are described in scha_calls(3HA).

optag Arguments

Macros that name resource type properties are listed below. The value of the named property of the resource's type is output.

Note – *optag* arguments, such as SCHA_API_VERSION and SCHA_BOOT, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* arguments.

SCHA_API_VERSION
The output argument is of type int*.

SCHA_BOOT
The output argument is of type char **.

SCHA_FAILOVER
The output argument is of type boolean_t *

SCHA_FINI
The output argument is of type char **.

SCHA_INIT
The output argument is of type char **.

SCHA_INIT_NODES
The output argument is of type scha_initnodes_flag_t *.

SCHA_INSTALLED_NODES
The output argument is of type scha_str_array_t **

SCHA_IS_LOGICAL_HOSTNAME
The output argument is of type boolean_t *

SCHA_IS_SHARED_ADDRESS
The output argument is of type boolean_t *.

SCHA_MONITOR_CHECK
The output argument is of type char **.

SCHA_MONITOR_START
The output argument is of type char **.

SCHA_MONITOR_STOP
The output argument is of type char **.

SCHA_PKGLIST
The output argument is of type scha_str_array_t **.

SCHA_POSTNET_STOP
The output argument is of type char **.

SCHA_PRENET_START
The output argument is of type char **.

SCHA_RESOURCE_LIST
The output argument is of type scha_str_array_t**

SCHA_RT_BASEDIR
The output argument is of type char **.

`SCHA_RT_DESCRIPTION`
The output argument is of type `char **`.

`SCHA_RT_SYSTEM`
The output argument is of type `boolean_t *`

`SCHA_RT_VERSION`
The output argument is of type `char **`.

`SCHA_SINGLE_INSTANCE`
The output argument is of type `boolean_t *`

`SCHA_START`
The output argument is of type `char **`.

`SCHA_STOP`
The output argument is of type `char **`.

`SCHA_UPDATE`
The output argument is of type `char **`.

`SCHA_VALIDATE`
The output argument is of type `char **`.

RETURN VALUES

The `scha_cluster_open()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scha_resource_get(1HA)`, `scha_calls(3HA)`, `scha_strerror(3HA)`, `attributes`(5)

NAME

scha_resourcetype_open, scha_resourcetype_close, scha_resourcetype_get – resource type information access functions.

SYNOPSIS

```
cc [flags...] -I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resourcetype_open(const char *rtname,
    scha_resourcetype_t *handle);

scha_err_t scha_resourcetype_close(scha_resourcetype_t handle);

scha_err_t scha_resourcetype_get(scha_resourcetype_t handle, const
    char *tag...);
```

DESCRIPTION

The `scha_resourcetype_open()`, `scha_resourcetype_get()`, and `scha_resourcetype_close()` functions are used together to access information on a resource type that is used by the Resource Group Manager (RGM) cluster facility.

`scha_resourcetype_open()` initializes access of the resource type and returns a handle to be used by `scha_resourcetype_get()`.

The *rtname* argument of `scha_resourcetype_open()` names the resource type to be accessed.

The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resourcetype_get()` accesses resource type information as indicated by the *tag* argument. The *tag* argument should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*.

An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved, or other information specific to the tag. The last argument in the argument list is to be of a type suitable type to hold the information indicated by *tag*. This is the "out" argument for the resource type information. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_resourcetype_get()` remains intact until `scha_resourcetype_close()` is called on the handle used for `scha_resourcetype_get()`.

`scha_resourcetype_close()` takes a *handle* argument returned from a previous call to `scha_resourcetype_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resourcetype_get()` calls that were made with the handle. Note that, memory, if needed to return a value, is allocated for each "get" call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resourcetype_get()` follow. The type of the output argument and any additional arguments are indicated. Structure and enum types are described in scha_calls(3HA).

optag Arguments

Macros that name resource type properties are listed below. The value of the named property of the resource's type is output.

Note – *optag* arguments, such as `SCHA_API_VERSION` and `SCHA_BOOT`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* arguments.

`SCHA_API_VERSION`
The output argument is of type `int*`.

`SCHA_BOOT`
The output argument is of type `char **`.

`SCHA_FAILOVER`
The output argument is of type `boolean_t *`

`SCHA_FINI`
The output argument is of type `char **`.

`SCHA_INIT`
The output argument is of type `char **`.

`SCHA_INIT_NODES`
The output argument is of type `scha_initnodes_flag_t *`.

`SCHA_INSTALLED_NODES`
The output argument is of type `scha_str_array_t **`

`SCHA_IS_LOGICAL_HOSTNAME`
The output argument is of type `boolean_t *`

`SCHA_IS_SHARED_ADDRESS`
The output argument is of type `boolean_t *`.

`SCHA_MONITOR_CHECK`
The output argument is of type `char **`.

`SCHA_MONITOR_START`
The output argument is of type `char **`.

`SCHA_MONITOR_STOP`
The output argument is of type `char **`.

`SCHA_PKGLIST`
The output argument is of type `scha_str_array_t **`.

`SCHA_POSTNET_STOP`
The output argument is of type `char **`.

`SCHA_PRENET_START`
The output argument is of type `char **`.

`SCHA_RESOURCE_LIST`
The output argument is of type `scha_str_array_t**`

`SCHA_RT_BASEDIR`
The output argument is of type `char **`.

`SCHA_RT_DESCRIPTION`
The output argument is of type `char **`.

`SCHA_RT_SYSTEM`
The output argument is of type `boolean_t *`

`SCHA_RT_VERSION`
The output argument is of type `char **`.

`SCHA_SINGLE_INSTANCE`
The output argument is of type `boolean_t *`

`SCHA_START`
The output argument is of type `char **`.

`SCHA_STOP`
The output argument is of type `char **`.

`SCHA_UPDATE`
The output argument is of type `char **`.

`SCHA_VALIDATE`
The output argument is of type `char **`.

RETURN VALUES

The `scha_cluster_open()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scha_resource_get(1HA)`, `scha_calls(3HA)`, `scha_strerror(3HA)`, `attributes`(5)

NAME

scha_resourcetype_open, scha_resourcetype_close, scha_resourcetype_get – resource type information access functions.

SYNOPSIS

```
cc [flags...] -I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

scha_err_t scha_resourcetype_open(const char *rtname,
    scha_resourcetype_t *handle);

scha_err_t scha_resourcetype_close(scha_resourcetype_t handle);

scha_err_t scha_resourcetype_get(scha_resourcetype_t handle, const
    char *tag...);
```

DESCRIPTION

The `scha_resourcetype_open()`, `scha_resourcetype_get()`, and `scha_resourcetype_close()` functions are used together to access information on a resource type that is used by the Resource Group Manager (RGM) cluster facility.

`scha_resourcetype_open()` initializes access of the resource type and returns a handle to be used by `scha_resourcetype_get()`.

The *rtname* argument of `scha_resourcetype_open()` names the resource type to be accessed.

The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_resourcetype_get()` accesses resource type information as indicated by the *tag* argument. The *tag* argument should be a string value defined by a macro in the `<scha_tags.h>` header file. Arguments following the tag depend on the value of *tag*.

An additional argument following the tag may be needed to indicate a cluster node from which the information is to be retrieved, or other information specific to the tag. The last argument in the argument list is to be of a type suitable type to hold the information indicated by *tag*. This is the "out" argument for the resource type information. No value is returned for the out parameter if the function fails. Memory that is allocated to hold information returned by `scha_resourcetype_get()` remains intact until `scha_resourcetype_close()` is called on the handle used for `scha_resourcetype_get()`.

`scha_resourcetype_close()` takes a *handle* argument returned from a previous call to `scha_resourcetype_open()`. It invalidates the handle and frees memory allocated to return values to `scha_resourcetype_get()` calls that were made with the handle. Note that, memory, if needed to return a value, is allocated for each "get" call. Space allocated to return a value in one call will not be overwritten and reused by subsequent calls.

Macros defined in `<scha_tags.h>` that may be used as *tag* arguments to `scha_resourcetype_get()` follow. The type of the output argument and any additional arguments are indicated. Structure and enum types are described in scha_calls(3HA).

optag Arguments

Macros that name resource type properties are listed below. The value of the named property of the resource's type is output.

Note – *optag* arguments, such as `SCHA_API_VERSION` and `SCHA_BOOT`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify *optag* arguments.

`SCHA_API_VERSION`
The output argument is of type `int*`.

`SCHA_BOOT`
The output argument is of type `char **`.

`SCHA_FAILOVER`
The output argument is of type `boolean_t *`

`SCHA_FINI`
The output argument is of type `char **`.

`SCHA_INIT`
The output argument is of type `char **`.

`SCHA_INIT_NODES`
The output argument is of type `scha_initnodes_flag_t *`.

`SCHA_INSTALLED_NODES`
The output argument is of type `scha_str_array_t **`

`SCHA_IS_LOGICAL_HOSTNAME`
The output argument is of type `boolean_t *`

`SCHA_IS_SHARED_ADDRESS`
The output argument is of type `boolean_t *`.

`SCHA_MONITOR_CHECK`
The output argument is of type `char **`.

`SCHA_MONITOR_START`
The output argument is of type `char **`.

`SCHA_MONITOR_STOP`
The output argument is of type `char **`.

`SCHA_PKGLIST`
The output argument is of type `scha_str_array_t **`.

`SCHA_POSTNET_STOP`
The output argument is of type `char **`.

`SCHA_PRENET_START`
The output argument is of type `char **`.

`SCHA_RESOURCE_LIST`
The output argument is of type `scha_str_array_t**`

`SCHA_RT_BASEDIR`
The output argument is of type `char **`.

`SCHA_RT_DESCRIPTION`
The output argument is of type `char **`.

`SCHA_RT_SYSTEM`
The output argument is of type `boolean_t *`

`SCHA_RT_VERSION`
The output argument is of type `char **`.

`SCHA_SINGLE_INSTANCE`
The output argument is of type `boolean_t *`

`SCHA_START`
The output argument is of type `char **`.

`SCHA_STOP`
The output argument is of type `char **`.

`SCHA_UPDATE`
The output argument is of type `char **`.

`SCHA_VALIDATE`
The output argument is of type `char **`.

RETURN VALUES

The `scha_cluster_open()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.

See `scha_calls(3HA)` for a description of other error codes.

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scha_resource_get(1HA)`, `scha_calls(3HA)`, `scha_strerror(3HA)`, `attributes`(5)

NAME

scha_strerror – map error code to error message

SYNOPSIS

```
cc [flags...]-I /usr/cluster/include file -L /usr/cluster/lib -l scha
#include <scha.h>

char *scha_strerror(scha_err_t err_code);
```

DESCRIPTION

The `scha_strerror()` routine translates the given `scha_err_t` error code to an appropriate, but terse, error message. The `char*` string returned by this routine is *not* internationalized, as its return value is to be used by the resource type implementation for logging to the system log facility, `syslog`(3C).

RETURN VALUES

The following return value is supported:

`const char` String describing the meaning of the error_code.

EXAMPLES

EXAMPLE 1 Using the `scha_strerror()` Routine

```
sample()
{
  scha_err_t  err;

 char * resource_group = "example_RG"; /* resource group containing example_R */
 char * resource_name = "example_R";   /* a configured resource */

 err = scha_control(SCHA_GIVEOVER, resource_group, resource_name);

 if (err != SCHA_ERR_NOERR) {
     syslog(LOG_ERR, "scha_control GIVEOVER failed: %s",
     scha_strerror(err));
 }
}
```

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scha_calls(3HA)`, `syslog`(3C), `attributes`(5)

SC31 4

NAME

clusters – cluster names database

SYNOPSIS

```
/etc/clusters
```

DESCRIPTION

The `clusters` file contains information regarding the known clusters in the local naming domain. For each cluster a single line should be present with the following information:

```
clustername        whitespace-delimited list of hosts
```

Expansion is recursive if a name on the right hand side is tagged with the expansion marker: "`*`".

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment. Characters up to the end of the line are not interpreted by routines which search the file.

Cluster names may contain any printable character other than an upper case character, a field delimiter, NEWLINE, or comment character. The maximum length of a cluster name is 32 characters.

This information is used by Sun Cluster system administration tools, like `cconsole(1M)` to specify a group of nodes to administer. The names used in this database must be host names, as used in the hosts database.

The database is available from either NIS or NIS+ maps or a local file. Lookup order can be specified in the `/etc/nsswitch.conf` file. The default order is nis files.

EXAMPLES

EXAMPLE 1 A Sample `/etc/clusters` File

Here is a typical `/etc/clusters` file:

```
bothclusters        *planets *wine
planets            mercury venus
wine                zinfandel merlot chardonnay riesling
```

Here is a typical `/etc/nsswitch.conf` entry:

```
clusters: nis files
```

FILES

`/etc/clusters`

`/etc/nsswitch.conf`

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu
Interface Stability	Uncommitted

SEE ALSO

`cconsole(1M)`, `chosts(1M)`, `serialports(4)`, `nsswitch.conf`(4), `attributes`(5)

NAME

rt_reg – resource type registration file

DESCRIPTION

The resource type registration file describes a resource type. Resource types represent highly-available or scalable services that run under the control of the Resource Group Manager (RGM) cluster facility. The file is part of a resource type implementation and is used as an input file for the `scrgadm(1M)` command to register the resource type into the cluster configuration. Registering the resource type is a prerequisite to creating resources of that type to run on the cluster.

A registration file declares the resource type properties and resource properties of a resource type. The file is divided into two parts, the declaration of resource type properties, and of resource properties. Note that property-names recognition is case insensitive.

The resource type property declarations provide the information on the resource type implementation, such as paths to the callback methods that are to be invoked by the RGM to control resources of the type. Most resource type properties have fixed values set in the `rt_reg` file. These properties are inherited by all resources of the type.

A resource type implementor can also customize and extend the administrative view of resource properties. There are two kinds of resource properties that can have entries in the second part of an `rt_reg` file: system defined properties and extension properties.

System-defined resource properties have predetermined types and semantics. The `rt_reg` file can be used to set attributes such as default, minimum and maximum values for system defined resource properties. The `rt_reg` file can also be used to declare extension properties that are defined entirely by the resource type implementation. Extension properties provide a way for a resource type to add information to the configuration data for a resource that is maintained and managed by the cluster system.

The `rt_reg` file can set default values for resource properties, but the actual values are set in individual resources. The properties in the `rt_reg` file can be variables that can be set to different values and adjusted by the cluster administrator.

Resource Type Property Declarations

The resource type property declarations consist of a number of property value assignments.

```
PROPERTY_NAME = "Value";
```

See the `rt_properties(5)` man page for a list of the resource type properties you can declare in the `rt_reg` file. Since most properties have default values or are optional, the only declarations that are essential in a resource type registration file are the type name, the paths to the `START` and `STOP` callback methods, and RT_version.

Note that the first property in the file must be the `Resource_type` property.

Starting in Sun Cluster 3.1, a resource type name is of the form

```
vendor_id.rtname:version
```

The three components of the resource type name are properties specified in the RTR file as *Vendor_id*, *Resource_type*, and *RT_version*; the `scrgadm` command inserts the period and colon delimiters. Although optional, the *Vendor_id* prefix is recommended to distinguish betweentwo registration files of the same name provided by different vendors. To ensure that the *Vendor_id* is unique, the recommended approach is to use the stock symbol for the company creating the resource type.

Resource type names created prior to Sun Cluster 3.1 continue to be of the form:

```
vendor_id.rtname
```

Resource Property Declarations

Resource property declarations consist of a number of entries, each entry being a bracketed list of attribute value assignments. The first attribute in the entry must be the resource property name.

System-defined properties have predetermined type and description attributes and so these attributes cannot be redeclared in the `rt_reg` file. Range restrictions, a default value, and constraints on when the value can be set by the administrator can be declared for system defined properties.

Attributes that can be set for system-defined properties are listed in the `property_attributes(5)` man page. Attributes not available for system-defined properties are noted as such in the table.

System-defined properties that can have entries in the `rt_reg` file are listed in the `r_properties(5)` man page. The following is a sample entry for the system defined `RETRY_COUNT` resource property.

```
{
  PROPERTY = RETRY_COUNT;
  MIN=0;
  MAX=10;
  DEFAULT=2;
  TUNABLE = ANYTIME;
}
```

Entries for extension properties must indicate a type for the property. Attributes that can be set for extension properties are listed in the `property_attributes(5)` man page.

The following is a sample entry for an extension property named "`ConfigDir`" that is of string type. The `TUNABLE` attribute indicates that the cluster administrator can set the value of the property when a resource is created.

```
{
  PROPERTY = ConfigDir;
  EXTENSION;
  STRING;
  DEFAULT="/";
  TUNABLE = AT_CREATION;
}
```

Usage An `rt_reg` file is an ASCII text file. It can include comments describing the contents of the file. The contents are the two parts described above, with the resource type property list preceding the resource property declarations.

White space can be blanks, tabs, newlines, or comments. White space can exist before or after tokens. Blanks and the pound sign (#) are not considered to be white space when found in quoted value tokens. White space separates tokens but is otherwise ignored.

Comments begin with # and end with the first newline encountered, inclusively.

Directives begin with #$ and end with the first newline encountered, inclusively. Directives must appear in the RTR file between the resource type property declaration section and the resource property declaration section. Directives inserted in any other location in the RTR file will produce parser errors. The only valid directives are `#$upgrade` and `#$upgrade_from`. Any other directive will produce parser errors.

Tokens are property names, property values, and the following:

`{ }` Encloses parameter table properties

`;` Terminates properties and attributes

`=` Separates property names and property values or attribute names and attribute values

`,` Separates values in a value list

The recognition of property-name keywords in the file is case insensitive.

Properties and attributes have one of three formats.

```
<property-name> = <property-value>;
<property-name>;
<property-name> = <property-value> [, <property-value>];
```

In the format above, the square brackets, `[ ]`, enclose optional items. That is, the property value can be a single `<property-value>` or a list of two or more `<property-value>`s separated by commas.

The first property in the property list must be the simple resource type name.

Boolean properties and attributes have the following syntax:

```
<boolean-property-name>;
<boolean-property-name> = TRUE;
<boolean-property-name> = FALSE;
```

The first and second forms both set the `<boolean-property-name>` to `TRUE`.

The only property name taking a list for its value is PKGLIST. An example is:

```
PKGLIST = SUNWscu, SUNWrsm;
```

Resource type property names are listed in the rt_properties(5) man page. System-defined properties are listed in the r_properties(5) man page.

Resource declarations consist of any number of entries, each being a bracketed list of resource property attributes.

```
{<attribute-value-list>}
```

Each attribute-value-list consists of attribute values for a resource property, in the same syntax used for property values, with the addition of the two type-attribute formats.

```
<type-attribute-value>;
<enum-type-attribute> { <enum-value> [ , <enum-value> ] };
```

The <type-attribute-value> syntax declares the data type of the extension property to have the value <type-attribute-value>. It differs from the first format of the <boolean-property-name>, which defines the property named by <boolean-property-name> to have the value TRUE.

For example, the TUNABLE attribute can have one of the following values: FALSE or NONE, AT_CREATION, TRUE or ANYTIME, and WHEN_DISABLED. When the TUNABLE attribute uses the syntax:

```
TUNABLE;
```

it gets the value of ANYTIME.

Grammar

The following is a description of the syntax of the rt_reg file with a BNF-like grammar. Non-terminals are in lower case, and terminal keywords are in upper case, although the actual recognition of keywords in the rt_reg file is case insensitive. The colon (:) following a non-terminal at the beginning of a lines indicates a grammar production. Alternative right-hand-sides of a grammar production are indicated on lines starting with a vertical bar (|). Variable terminal tokens are indicated in angled brackets and comments are parenthesized. Other punctuation in the right-hand side of a grammar production, such as semi-colon (;), equals sign (=), and angled brackets ({ }) are literals.

A comment has the form:

```
COMMENT    : # <anything but NEWLINE> NEWLINE
```

Comments may appear after any token. Comments are treated as white-space.

```
    rt_reg_file    :  Resource_type = value ; proplist paramtable

    proplist    :  (NONE: empty)
```

```
| proplist rtproperty

rtproperty    : rtboolean_prop ;
| rtvalue_prop ;

rtboolean_prop    : SINGLE_INSTANCE
| FAILOVER | RT_SYSTEM

rtvalue_prop    : rtprop = value
| PKGLIST = valuelist

rtprop    : RT_BASEDIR
| RT_VERSION
| API_VERSION
| INIT_NODES
| START
| STOP
| VALIDATE
| UPDATE
| INIT
| FINI
| BOOT
| MONITOR_START
| MONITOR_STOP
| MONITOR_CHECK
| PRENET_START
| POSTNET_STOP
| RT_DESCRIPTION
| VENDOR_ID
| rtboolean_prop (booleans may have explicit assignments.)

value    : <contiguous-non-ws-non-;-characters>
| "<anything but quote>"
| TRUE
| FALSE
| ANYTIME
| WHEN_DISABLED
| AT_CREATION
| RG_PRIMARIES
| RT_INSTALLED_NODES
|  (NONE: Empty value)

valuelist    : value
| valuelist , value

upgradesect : (empty)
| #$UPGRADE upgradelist

upgradelist :  (empty)
| upgradelist #$UPGRADE_FROM rt_version upgtunability

upgtunability : ANYTIME
| AT_CREATION
| WHEN_DISABLED
| WHEN_OFFLINE
| WHEN_UNMANAGED
| WHEN_UNMONITORED
```

```
paramtable    :  (empty)
| paramtable parameter

parameter    :  { pproplist }

pproplist    : PROPERTY = value ;  (property name must come first)
| pproplist pproperty

pproperty    : pboolean_prop ;
| pvalue_prop ;
| typespec ;

pvalue_prop    : tunable_prop
| pprop = value
| pprop =  (NONE: no value setting)
| DEFAULT = valuelist

pprop    : DESCRIPTION
| MIN
| MAX
| MINLENGTH
| MAXLENGTH
| ARRAY_MINSIZE
| ARRAY_MAXSIZE
| pboolean_prop

tunable_prop    : TUNABLE
| TUNABLE = AT_CREATION
| TUNABLE = ANYTIME
| TUNABLE = WHEN_DISABLED
| TUNABLE = TRUE
| TUNABLE = FALSE
| TUNABLE = NONE

typespec    : INT
| BOOLEAN
| STRING
| STRINGARRAY
| ENUM { valuelist }
```

EXAMPLES

EXAMPLE 1 A Sample Registration File

The following is the registration file for a simple example resource type.

```
#
# Registration information for example resource type
#

Resource_type = example_RT;
Vendor_id = SUNW;
RT_Version = 2.0
RT_Basedir= /opt/SUNWxxx;
START = bin/example_service_start;
STOP  = bin/example_service_stop;
Pkglist = SUNWxxx;
```

EXAMPLE 1 A Sample Registration File *(Continued)*

```
#$upgrade
#$upgrade_from "1.0" when_unmonitored

#
# Set range and defaults for method timeouts and Retry_count.
#
{ Property = START_TIMEOUT;  Tunable; MIN=60; DEFAULT=300; }
{ Property = STOP_TIMEOUT;  Tunable; MIN=60; DEFAULT=300; }
{ Property = Retry_count;  Tunable; MIN=1; MAX=20; DEFAULT=10; }

#
# An extension property that can be set at resource creation
#
{ Property = LogLevel;
  Extension;
  enum { OFF, TERSE, VERBOSE };
  Default = TERSE;
  Tunable = AT_CREATION;
  Description = "Controls the detail of example_service logging";
}
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO

`scrgadm`(1M), `attributes`(5), `rt_properties`(5), `r_properties`(5), `property_attributes`(5)

Sun Cluster 3.1 Data Services Developer's Guide

NAME

serialports – name to serial port database

SYNOPSIS

```
/etc/serialports
serialports NIS or NIS+ maps
```

DESCRIPTION

The `serialports` database maps a name to a server name and TCP port number that represents the serial port connected to the specified terminal server host. The database is typically used to map host names to their consoles, but may also be used to provide access to printers, modems, and the like. The mapping is used when the service is being provided by a network based terminal concentrator such as a Xylogics Annex or MicroAnnex. For each name a single line should be present with the following information:

```
host-name    concentrator-hostname tcp-port-number
```

Items are separated by any number of blanks or TAB characters. A '#' indicates the beginning of a comment. Characters after the hash up to the end of the line are not interpreted by routines that search the file.

This information is used by `cconsole(1M)` to establish connection to a group of consoles of a cluster of network hosts. The names used in this database must be host names, as used in the hosts database.

For E10000 nodes, the entries are different. This is because E10000 uses netcon for console purposes, which operates over a network and executes on the SSP. The following is the generic format for the entry.

```
<hostname> <SSPname> 23
```

The database is available from either the NIS or NIS+ maps or a local file. Lookup order is specified by the `serialports` entry in the `/etc/nsswitch.conf` file, if present. If no search order is specified, the default order is `nis files`.

EXAMPLES

EXAMPLE 1 A Sample `/etc/serialports` File

The following is an example `/etc/serialports` file:

```
# Network host to port database

#  NFS server cluster
mercury        planets-tc    5001
venus        planets-tc    5002

# E10000 server cluster
cashews         nuts-ssp-1 23
pecans          nuts-ssp-2 23
```

EXAMPLE 2 A Sample `/etc/nsswitch.conf` File Entry

The following is a typical `/etc/nsswitch.conf` entry:

```
serialports: nis files
```

FILES

`/etc/serialports`

`/etc/nsswitch.conf`

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Uncommitted

SEE ALSO

`cconsole`(1M), `chosts`(1M), `cports`(1M), `clusters`(4), `nsswitch.conf`(4), `attributes`(5)

SC31 5

NAME

SUNW.HAStorage, HAStorage – resource type to synchronize action between HA storage and data services

DESCRIPTION

`SUNW.HAStorage` describes a resource type that defines resources in a resource group to synchronize the actions between the cluster file system, global devices, and relevant data services.

There is no direct synchronization between resource groups and disk device groups (and the cluster file system). As a result, during a cluster reboot or failover, an attempt to start a data service can occur while its dependent global devices or cluster file systems are still unavailable. Consequently, the data service's `START` method might timeout and the service is not started on the cluster.

`SUNW.HAStorage` is a resource type that specifically monitors the storage device services. You add a resource of this type to resource groups containing other resources and set up dependencies between the other resources and the `HAStorage` resource. The `HAStorage` resource continually tests the availability of the global devices, device groups, and the cluster file system. The dependencies ensure that the data service resources does not attempt to start until the device services are available.

When a data service resource is set up with a "strong dependency" upon a `SUNW.HAStorage` resource, the data service resources are not started before all dependent global devices and cluster file systems become available.

Multiple `SUNW.HAStorage` resources can be set up within a cluster to obtain finer granularity of the service monitoring checks. Device services that the data service needs to check and wait for but not depend upon to be online can be defined in a separate resource, and a "weak dependency" can be set up from the data resource to the device resource.

In this case, the data service resource waits for the resource to check if the device services are all available. If not, even if the `SUNW.HAStorage` `START` method times out, the data service can still be brought online. This feature is useful to some data services. For example, assume a Web server depends on ten cluster file systems. If only one file system isn't ready within the timeout period, the Web service should still go online since it still can provide 90 percent of the services.

Two extension properties are associated with the `SUNW.HAStorage` resource type: `ServicePaths` and `AffinityOn`.

`ServicePaths` Contains valid global device group names, paths to global devices, or cluster file system mount points that are to be checked. They are defined in the format of

paths`[,...]`.

A typical example of a global device group is `nfs-dg`. A path to a global device is a valid device path in the global device namespace, such as `/dev/global/dsk/d5s2`, `/dev/global/dsk/d1s2`, or `/dev/global/rmt/0`. A cluster file system mount point is a valid global mount point defined in `/etc/vfstab` on all cluster nodes

of the cluster. You can define a global device group, a global device path, and a cluster file system mount point in one `SUNW.HAStorage` resource.

`AffinityOn` A boolean flag that specifies whether the `SUNW.HAStorage` resource needs to do an affinity switchover for the global devices and cluster file systems defined in `ServicePaths`.

When `AffinityOn` is set to `False`, the `SUNW.HAStorage` resource passively waits for the specified global services to become available. As a result, the primary of each online global service might not be the same node that is the primary of the resource group.

The purpose of an affinity switchover is to enhance performance by having data services and their dependent global services run on the same node. For each global service, the `SUNW.HAStorage` resource attempts affinity switchover only once. If switchover fails, nothing is affected and the availability check occurs normally.

The default value for `ServicePaths` is the empty string. The default value for `AffinityOn` is `True`. Both extension properties can be changed at any time when the resource group is offline.

For scalable service resources, the setting of the `AffinityOn` flag is ignored and no affinity switchover can be done. There is no benefit to switching over the disk device services because the scalable data service can be running on multiple nodes simultaneously.

SEE ALSO `rt_reg(4)`

NOTES `SUNW.HAStorage` specifies resources that check and wait for the specified global devices, device group, and cluster file systems to become available. The checking is only meaningful when data service resources (application resources) in the same resource group are set up with the correct dependency upon the `SUNW.HAStorage` resources. Otherwise, no synchronization is done.

Avoid configuring two different `SUNW.HAStorage` resources in different resource groups with their ServicePaths property referencing the same global resource and with both `AffinityOn` flags set to `True`. When the cluster is booting or during a switchover, the resource groups might end up mastered on two different nodes. Both of the `SUNW.HAStorage` resources would attempt to do an affinity switchover of the same device group, resulting in a race condition. In this case, redundant switchovers would occur and the device group might not end up being mastered by the most preferred node.

The waiting time for global services to become available is specified by the `Prenet_Start_Timeout` property in `SUNW.HAStorage`. The time is tunable with a default value of 30 minutes (1,800 seconds).

NAME

property_attributes – resource property attributes

DESCRIPTION

The list below describes the resource property attributes that can be used to change system-defined properties or create extension properties.

You cannot specify `NULL` or the empty string (`""`) as the default value for boolean, enum, or int types.

`Property`
The name of the resource property.

`Extension`
If used, indicates that the RTR file entry declares an extension property defined by the resource type implementation. Otherwise, the entry is a system-defined property.

`Description`
A string annotation intended to be a brief description of the property. The description attribute cannot be set in the RTR file for system-defined properties.

Property Type
Allowable types are: string, boolean, int, enum, and stringarray. You cannot set the type attribute in an RTR file entry for system-defined properties. The type determines acceptable property values and the type-specific attributes that are allowed in the RTR file entry. An enum type is a set of string values.

`Default`
Indicates a default value for the property.

`Tunable`
Indicates when the cluster administrator can set the value of this property in a resource. Can be set to `None` or `False` to prevent the administrator from setting the property. Values that allow administrator tuning are: `True` or `Anytime` (at any time), `At_creation` (only when the resource is created), or `When_disabled` (when the resource is offline).

The default is True (Anytime).

`Enumlist`
For an enum type, a set of string values permitted for the property.

`Min`
For an int type, the minimal value permitted for the property. Note that you cannot specify Min=0 for a method timeout.

`Max`
For an int type, the maximum value permitted for the property. Note that you cannot specify a maximum value for a method timeout.

`Minlength`
For string and stringarray types, the minimum string length permitted.

`Maxlength`
For string and stringarray types, the maximum string length permitted.

`Array_minsize`
For stringarray type, the minimum number of array elements permitted.

`Array_maxsize`
For stringarray type, the maximum number of array elements permitted.

EXAMPLES

EXAMPLE 1 An `int` Type Definition

An int type definition might look like this:

```
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 30;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Time out value for the probe (seconds)";
}
```

SEE ALSO

`scrgadm`(1M), `r_properties`(5) `rg_properties`(5), `rt_properties`(5)

NAME

SUNW.rac_cvm, rac_cvm – resource type implementation that represents the VERITAS Volume Manager (VxVM) component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_cvm` resource type represents the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can use the `SUNW.rac_cvm` resource type to represent this component *only* if the cluster feature of VxVM is enabled.

Instances of the `SUNW.rac_cvm` resource type hold VxVM component configuration parameters. Instances of this type also show the status of a reconfiguration of the VxVM component.

The `SUNW.rac_cvm` resource type is a single-instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup(1M)` utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm(1M)` utility

You can set the following extension properties of the VxVM component resource by using the `scrgadm` utility.

Note – Some extension properties are tunable only when the resource is disabled. You can modify such extension properties only when VxVM is *not* running in cluster mode on any cluster node.

`Cvm_abort_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 40. This property specifies the timeout (in seconds) for the abort step of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_return_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 40. This property specifies the timeout (in seconds) for the return step of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_start_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the start step of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_step1_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 1 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_step2_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 2 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_step3_timeout`
Type integer; minimum 30; maximum 99999; defaults to 240. This property specifies the timeout (in seconds) for step 3 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_step4_timeout`
Type integer; minimum 100; maximum 99999; defaults to 320. This property specifies the timeout (in seconds) for step 4 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_stop_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 40. This property specifies the timeout (in seconds) for the stop step of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Reservation_timeout`
Type integer; minimum 100; maximum 99999; defaults to 325. This property specifies the timeout (in seconds) for the reservation step of a reconfiguration of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Vxclust_num_ports`
Type integer; minimum 16; maximum 64; defaults to 32. This property specifies the number of communications ports that the `vxclust` program uses. You can modify this property only when the resource is disabled. The modified value is used for the next reconfiguration of the VxVM component.

`Vxclust_port`
Type integer; minimum 1024; maximum 65535; defaults to 5568. This property specifies the communications port number that the `vxclust` program uses. You can modify this property only when the resource is disabled. The modified value is used for the next reconfiguration of the VxVM component.

`Vxconfigd_port`
Type integer; minimum 1024; maximum 65535; defaults to 5560. This property specifies the communications port number that the VxVM component configuration daemon `vxconfigd` uses. You can modify this property only when the resource is disabled. The modified value is used for the next reconfiguration of the VxVM component.

`Vxkmsgd_port`
Type integer; minimum 1024; maximum 65535; defaults to 5559. This property specifies the communications port number that the VxVM component messaging daemon `vxkmsgd` uses. You can modify this property only when the resource is disabled. The modified value is used for the next reconfiguration of the VxVM component.

EXAMPLES

EXAMPLE 1 Changing a Property of a `rac_cvm` Resource

This example sets the timeout for step 4 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters to 300 seconds. The example assumes that an instance of the `SUNW.rac_cvm` resource type named `rac_cvm` has been created.

```
example# scrgadm -c -j rac_cvm\\
         -x cvm_step4_timeout=300
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcvm

SEE ALSO

`scrgadm`(1M), `scsetup`(1M), `attributes`(5)

NAME

SUNW.rac_framework, rac_framework – resource type implementation for the framework that enables Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_framework` resource type represents the framework that enables Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. This resource type enables you to monitor the status of this framework.

The `SUNW.rac_framework` resource type is a single instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup`(1M) utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm`(1M) utility

The Sun Cluster Support for Oracle Parallel Server/Real Application Clusters framework resource has no extension properties.

EXAMPLES

EXAMPLE 1 Creating a `rac_framework` Resource

This example registers the `SUNW.rac_framework` resource type and creates an instance of the `SUNW.rac_framework` resource type named `rac_framework`. The example assumes that a resource group named `rac-framework-rg` has been created.

```
example# scrgadm -a -t SUNW.rac_framework
example# scrgadm -a -j rac_framework \
         -g rac-framework-rg \
         -t SUNW.rac_framework
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscucm

SEE ALSO

`scrgadm`(1M), `scsetup`(1M), `attributes`(5)

NAME

SUNW.rac_hwraid, rac_hwraid – resource type implementation that represents the hardware redundant array of independent disks (RAID) component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_hwraid` resource type represents the hardware RAID component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters.

The `SUNW.rac_hwraid` resource type is a single-instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup(1M)` utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm(1M)` utility

You can set the following extension properties of the hardware RAID resource by using the `scrgadm` utility.

`Reservation_timeout`
Type integer; minimum 100; maximum 99999; defaults to 325. This property specifies the timeout (in seconds) for the reservation step of a reconfiguration of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

EXAMPLES

EXAMPLE 1 Changing a Property of a `rac_hwraid` Resource

This example sets the timeout for the reservation step of a reconfiguration of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters to 350 seconds. The example assumes that an instance of the `SUNW.rac_hwraid` resource type named `rac_hwraid` has been created.

```
example# scrgadm -c -j rac_hwraid\
         -x reservation_timeout=350
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWhwraid

SEE ALSO

`scrgadm(1M)`, `scsetup(1M)`, `attributes`(5)

NAME

SUNW.rac_svm, rac_svm – resource type implementation that represents the Solaris Volume Manager component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_svm` resource type represents the Solaris Volume Manager for Sun Cluster component of the Sun Cluster framework for Oracle Parallel Server/Real Application Clusters.

Instances of the `SUNW.rac_svm` resource type hold Solaris Volume Manager for Sun Cluster component configuration parameters. Instances of this type also show the status of a reconfiguration of the Solaris Volume Manager for Sun Cluster component.

The `SUNW.rac_svm` resource type is a single-instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup(1M)` utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm(1M)` utility

You can set the following extension properties of the Solaris Volume Manager for Sun Cluster component resource by using the `scrgadm` utility.

`Debug_level`
Type integer; minimum 0; maximum 10; defaults to 1. This property specifies the debug level for the Solaris Volume Manager for Sun Cluster module of Sun Cluster framework for Oracle Parallel Server/Real Application Clusters. When the debug level is increased, more messages are written to the log files during reconfiguration. You can modify this property at any time.

`Reservation_timeout`
Type integer; minimum 100; maximum 99999; defaults to 325. This property specifies the timeout (in seconds) for the reservation step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_abort_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the abort step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_return_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the return step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_start_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the start step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_step1_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for step 1 of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_step2_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for step 2 of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_step3_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for step 3 of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_step4_timeout`
Type integer; minimum 100; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for step 4 of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_stop_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the stop step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

EXAMPLES

EXAMPLE 1 Changing a Property of a `rac_svm` Resource

This example sets the timeout for step 4 of a reconfiguration of the Solaris Volume Manager for Sun Cluster component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters to 300 seconds. The example assumes that an instance of the `SUNW.rac_svm` resource type named `rac_svm` has been created.

```
example# scrgadm -c -j rac_svm \
         -x svm_step4_timeout=300
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWscmd

SEE ALSO `attributes`(5)

`scrgadm`(1M), `scsetup`(1M)

NAME

SUNW.rac_udlm, rac_udlm – resource type implementation for the configuration of the UNIX Distributed Lock Manager (Oracle UDLM) component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_udlm` resource type enables the management of the Oracle UDLM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. The management of this component involves the following activities:

- Setting the parameters of the Oracle UDLM component
- Monitoring the status of the Oracle UDLM component

The `SUNW.rac_udlm` resource type is a single-instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup(1M)` utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm(1M)` utility

You can set the following extension properties for an Oracle UDLM resource by using the `scrgadm` utility.

Note – Some extension properties are tunable only when the resource is disabled. You can modify such extension properties only when the Oracle UDLM is *not* running on any cluster node.

`Failfastmode`
Type enum; defaults to `panic`. This property specifies the failfast mode of the node on which the Oracle UDLM is running. The failfast mode determines the action that is performed in response to a critical problem with this node. The possible values of this property are as follows:

`off`	Failfast mode is disabled.
`panic`	The node is forced to panic.

You can modify this property at any time. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Num_ports`
Type integer; minimum 16; maximum 64; defaults to 32. This property specifies the number of communications ports that the Oracle UDLM uses. You can modify this property only when the resource is disabled. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Oracle_config_file`
Type string; defaults to `/etc/opt/SUNWcluster/conf/udlm.conf`. This property specifies the configuration file that the Oracle distributed lock manager

(DLM) uses. This file must already exist. The file is installed when the Oracle software is installed. For more information, refer to the documentation for the Oracle software. You can modify this property at any time. The modified value is used for the next start-up of the Oracle DLM.

`Port`
Type integer; minimum 1024; maximum 65500; defaults to 6000. This property specifies the communications port number that the Oracle UDLM uses. You can modify this property only when the resource is disabled. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Schedclass`
Type enum; defaults to `RT`. This property specifies the scheduling class of the Oracle UDLM that is passed to the `priocntl`(1) command. The possible values of this property are as follows:

`RT`	Real-time
`TS`	Time-sharing
`IA`	Interactive

You can modify this property only when the resource is disabled. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Schedpriority`
Type integer; minimum 0; maximum 59; defaults to 11. This property specifies the scheduling priority of the Oracle UDLM that is passed to the `priocntl` command. You can modify this property only when the resource is disabled. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Udlm_abort_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 325. This property specifies the timeout (in seconds) for the abort step of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_start_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for the start step of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Udlm_step1_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 1 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_step2_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 2 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_step3_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 3 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_step4_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 4 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_step5_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 5 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

EXAMPLES

EXAMPLE 1 Changing a Property of a `rac_udlm` Resource

This example sets the timeout for step 4 of a reconfiguration of the Oracle UDLM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters to 45 seconds. The example assumes that an instance of the `SUNW.rac_udlm` resource type named `rac_udlm` has been created.

```
example# scrgadm -c -j rac_udlm\
         -x udlm_step4_timeout=45
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWudlm

SEE ALSO

`priocntl`(1), `scrgadm`(1M), `scsetup`(1M), `attributes`(5)

NAME

SUNW.RGOffload, RGOffload – resource type to offload specified resource groups

DESCRIPTION

`SUNW.RGOffload` describes a resource type that allows resources configured in failover resource groups to offload other specified resource groups.

This facility is most useful when the limited resources on cluster nodes prevent multiple data services from running simultaneously on a node. In such situations, a `RGOffload` resource in a resource group containing critical data services is configured to offload other resource groups.

You can use the `scrgadm(1M)` command or resource configuration GUI to add a `RGOffload` resource to the resource group containing critical data service resources, setup dependencies of the critical data service resources on this resource, and configure the resource groups to be offloaded from a node when critical data service resources are running on it. The dependencies ensure that the data service resources do not attempt to start on a node until the `START` method of the `RGOffload` resource has offloaded, or at least attempted to offload the specified resource groups from the node.

Resource groups specified to be offloaded must have their Desired_primaries property set to 0. The fault monitor of the `SUNW.RGOffload` resource will attempt to keep such resource groups online on as many healthy nodes as possible, limited by the Maximum_primaries property of individual resource groups. The fault monitor checks the status of specified resource groups on all nodes every `Thorough_probe_interval`.

When a data service resource is set up with a "strong dependency" upon a `SUNW.RGOffload` resource, the data service resource is not started on a node if there is a failure in offloading specified resource groups from that node. A data service resource set up with a "weak dependency" upon the `SUNW.RGOffload` resource may start when specified resource groups cannot be successfully offloaded from the node. An attempt would be made to offload the specified resource groups, but a failure in doing so will not prevent the startup of the data service resource.

See `r_properties(5)` for a complete description of the standard resource properties.

Extension Properties

`Monitor_retry_count`
Type integer; defaults to `4`. This property controls fault-monitor restarts. The property indicates the number of times that the process monitor facility (PMF) restarts the fault monitor. The property corresponds to the `-n` option passed to the `pmfadm(1M)` command. The RGM counts the number of restarts in a specified time window (see the property `Monitor_retry_interval`). Note that this property refers to the restarts of the fault monitor itself, not the `SUNW.RGOffload` resource. You can modify the value for this property at any time.

`Monitor_retry_interval`
Type integer; defaults to `2`. This property indicates the time window in minutes during which the RGM counts fault-monitor failures. The property corresponds to the `-t` option passed to the `pmfadm(1M)` command. If the number of times that the fault monitor fails exceeds the value of the extension property

Monitor_retry_count, the PMF does not restart the fault monitor. You can modify the value for this property at any time.

rg_to_offload
: Type string array, specified as a comma-separated list of resource groups. No default exists for this field. You must provide the value when creating the resource. This property indicates the list of resource groups to be offloaded. All resource groups in this property must have Desired_primaries set to 0. rg_to_offload should not contain the resource group in which the RGOffload resource is being configured. rg_to_offload should also not contain resource groups dependent upon each other. For example, if resource group RG-B depends on resource group RG-A, then both, RG-A and RG-B should not be configured in this extension property. SUNW.RGOffload resource type does not check for dependencies among resource groups in the rg_to_offload extension property. You can modify the value of this property at any time.

continue_to_offload
: Type boolean; defaults to TRUE. This property indicates whether to continue offloading the next resource group in the list specified in the rg_to_offload property in case of error in offloading any resource group. You can modify the value of this property at any time.

max_offload_retry
: Type integer; defaults to 15. This property indicates the number of attempts during the startup of RGOffload resource to offload a resource group specified in the rg_to_offload property if there is a failure due to cluster or resource group reconfiguration. This value applies to all resource groups in the rg_to_offload property. When the value of this property is greater than 0, successive attempts to offload the same resource group would be made after approximately 10 second intervals. You can modify the value of this property at any time.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWrgofl

SEE ALSO

pmfadm(1M), scha_resource_get(1HA), scrgadm(1M), scswitch(1M), scha_cluster_get(3HA), scha_resourcegroup_get(3HA) , attributes(5), r_properties(5)

Sun Cluster Data Services Installation and Configuration Guide

NAME

rg_properties – resource group properties

DESCRIPTION

The list below describes the resource group properties that are defined by Sun Cluster.

Resource Group Properties and Descriptions

Note – Resource group property names, such as `Auto_start_on_new_cluster` and `Desired_primaries`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify resource group property names.

`Auto_start_on_new_cluster` (`boolean`)
This property controls whether the Resource Group Manager (RGM) starts the resource group automatically when a new cluster is forming. The default is `TRUE`.

If set to `TRUE`, the RGM attempts to start the resource group automatically to achieve `Desired_primaries` when all the nodes of the cluster are simultaneously rebooted.

If set to `FALSE`, the resource group does not start automatically when the cluster is rebooted. The resource group remains offline until the first time that the resource group is manually switched online by using `scswitch(1M)` or the equivalent graphical user interface command. After that, the resource group resumes normal failover behavior.

Default `TRUE`

Tunable Any time

`Desired_primaries` (`integer`)
The desired number of nodes that the group can run on simultaneously.

The default is 1. If the `RG_mode` property is `Failover`, the value of this property must be no greater than 1. If the `RG_mode` property is `Scalable`, a value greater than 1 is allowed.

Default 1, see above

Tunable Any time

`Failback` (`boolean`)
A Boolean value that indicates whether to recalculate the set of nodes where the group is online when the cluster membership changes. A recalculation can cause the RGM to bring the group offline on less preferred nodes and online on more preferred nodes.

Default `FALSE`

Tunable Any time

`Global_resources_used` (`string_array`)
Indicates whether cluster file systems are used by any resource in this resource group. Legal values that the administrator can specify are an asterisk (*) to indicate all global resources, and the empty string (“”) to indicate no global resources.

Default All global resources

Tunable Any time

`Implicit_network_dependencies` (`boolean`)
A Boolean value that indicates, when `TRUE`, that the RGM should enforce implicit strong dependencies of non-network-address resources on network-address resources within the group. This means that the RGM starts all network-address resources before all other resources and stops network address resources after all other resources within the group. Network-address resources include the logical host name and shared address resource types.

In a scalable resource group, this property has no effect because a scalable resource group does not contain any network-address resources.

Default `TRUE`

Tunable Any time

`Maximum_primaries` (`integer`)
The maximum number of nodes where the group might be online at once.

If the `RG_mode` property is `Failover`, the value of this property must be no greater than 1. If the `RG_mode` property is `Scalable`, a value greater than 1 is allowed.

Default 1, see above

Tunable Any time

`Nodelist` (`string_array`)
A list of cluster nodes where the group can be brought online in order of preference. These nodes are known as the potential primaries or masters of the resource group.

Default The list of all cluster nodes in arbitrary order

Tunable Any time

`Pathprefix` (`string`)
A directory in the cluster file system in which resources in the group can write essential administrative files. Some resources might require this property. Make `Pathprefix` unique for each resource group.

Default The empty string

Tunable Any time

`Pingpong_interval` (`integer`)
A non-negative integer value (in seconds) used by the RGM to determine where to bring the resource group online in the event of a reconfiguration or as the result of an `scha_control` giveover command or function being executed.

In the event of a reconfiguration, if the resource group fails more than once to come online within the past `Pingpong_interval` seconds on a particular node (because the resource's `Start` or `Prenet_start` method exited nonzero or timed out), that node is considered ineligible to host the resource group and the RGM looks for another master.

If a scha_control(1HA) command or scha_control(3HA) giveover is executed on a given node by a resource, thereby causing its resource group to fail over to another node, the first node (on which scha_control was invoked) cannot be the destination of another scha_control giveover by the same resource until Pingpong_interval seconds have elapsed.

Default 3600 (one hour)

Tunable Any time

Resource_list (string_array)
The list of resources that are contained in the group. The administrator does not set this property directly. Rather, the RGM updates this property as the administrator adds or removes resources from the resource group.

Default No default

Tunable Never

RG_affinities (string)
The RGM is to try to locate a resource group on a node that is a current master of another given resource group (positive affinity), or to locate a resource group on a node that is not a current master of a given resource group (negative affinity).

You can set RG_affinities to the following strings:

- ++, or strong positive affinity
- +, or weak positive affinity
- -, or weak negative affinity
- --, or strong negative affinity
- +++, or strong positive affinity with failover delegation For example, RG_affinities=+RG2,--RG3 indicates that this resource group has a weak positive affinity for RG2 and a strong negative affinity for RG3.

Using RG_affinities is described in "Administering Data Service Resources" in *Sun Cluster Data Services Planning and Administration Guide for Solaris OS*.

Default The empty string

Tunable Any time

RG_dependencies (string_array)
Optional list of resource groups that indicate a preferred ordering for bringing other groups online or offline on the same node. The graph of all strong RG_affinities (positive and negative) together with RG_dependencies is not allowed to contain cycles.

For example, suppose that resource group RG2 is listed in the RG_dependencies list of resource group RG1. In other words, suppose that RG1 has a resource group dependency on RG2. The following list summarizes the effects of this resource group dependency:

- When a node joins the cluster, `Boot` methods on that node are not run on resources in RG1 until all `Boot` methods on that node have completed on resources in RG2.
- If RG1 and RG2 are both in the `PENDING_ONLINE` state on the same node at the same time, the starting methods (`Prenet_start` or `Start`) are not run on any resources in RG1 until all the resources in RG2 have completed their starting methods.
- If RG1 and RG2 are both in the `PENDING_OFFLINE` state on the same node at the same time, the stopping methods (`Stop` or `Postnet_stop`) are not run on any resources in RG2 until all the resources in RG1 have completed their stopping methods.
- An attempt to switch the primaries of RG1 or RG2 fails if switching the primaries would leave RG1 online on any node and RG2 offline on all nodes. `scswitch`(1M) and `scsetup`(1M) contain more information.
- Setting the `Desired_primaries` property to a value that is greater than zero on RG1 is not permitted if `Desired_primaries` is set to zero on RG2.
- Setting the `Auto_start_on_new_cluster` property to `TRUE` on RG1 is not permitted if `Auto_start_on_new_cluster` is set to `FALSE` on RG2.

Default The empty list

Tunable Any time

`RG_description` (`string`)
A brief description of the resource group.

Default The empty string

Tunable Any time

`RG_is_frozen` (`boolean`)
A Boolean value that indicates whether a global device on which a resource group depends is being switched over. If this property is set to `TRUE`, the global device is being switched over. If this property is set to `FALSE`, no global device is being switched over. A resource group depends on global devices as indicated by its `Global_resources_used` property.

You do not set the `RG_is_frozen` property directly. The RGM updates the `RG_is_frozen` property when the status of the global devices changes.

Default No default

Tunable Never

`RG_mode` (`enum`)
Indicates whether the resource group is a failover or a scalable group. If the value is `Failover`, the RGM sets the `Maximum_primaries` property of the group to 1 and restricts the resource group to being mastered by a single node.

If the value of this property is `Scalable`, the RGM allows the `Maximum_primaries` property to be set to a value that is greater than 1. As a result, the group can be mastered by multiple nodes simultaneously. The RGM does not allow a resource whose `Failover` property is `TRUE` to be added to a resource group whose `RG_mode` is `Scalable`.

If `Maximum_primaries` is 1, the default is `Failover`. If `Maximum_primaries` is greater than 1, the default is `Scalable`.

Default	Depends on the value of `Maximum_primaries`
Tunable	At creation

`RG_name` (`string`)
The name of the resource group. This property is required and must be unique within the cluster.

Default	No default
Tunable	At creation

`RG_project_name` (`string`)
The Solaris project name (see `projects`(1)) that is associated with the resource group. Use this property to apply Solaris resource management features, such as CPU shares and resource pools, to cluster data services. When the RGM brings resource groups online, it launches the related processes under this project name for resources that do not have the `Resource_project_name` property set (see `r_properties(5)`). The specified project name must exist in the projects database (see `projects`(1) and *System Administration Guide: Resource Management and Network Services*).

This property is only supported starting in Solaris 9.

Note – Changes to this property take affect the next time that the resource is started.

Default	The text string "`default`"
Tunable	Any time
Valid value	Any valid Solaris project name

`RG_state` on each cluster node (`enum`)
Set by the RGM to `UNMANAGED`, `ONLINE`, `OFFLINE`, `PENDING_ONLINE`, `PENDING_OFFLINE`, `ERROR_STOP_FAILED`, `ONLINE_FAULTED`, or `PENDING_ONLINE_BLOCKED` to describe the state of the group on each cluster node.

You cannot configure this property. However, you can indirectly set this property by invoking `scswitch`(1M), or by using the equivalent `scsetup`(1M) or SunPlex Manager commands. A group can exist in an `UNMANAGED` state when that group is not under the control of the RGM.

The following descriptions summarize each state.

Note – States apply to individual nodes only, except the `UNMANAGED` state, which applies across all nodes. For example, a resource group might be `OFFLINE` on node A, but `PENDING_ONLINE` on node B.

`UNMANAGED`	The initial state of a newly created resource group, or the state of a previously managed resource group. Either `Init` methods have not yet been run on resources in the group, or `Fini` methods have been run on resources in the group. The group is not managed by the RGM.
`ONLINE`	The resource group has been started on the node. In other words, the starting methods (`Prenet_start`, `Start`, and `Monitor_start`, as applicable to each resource) have executed successfully on all enabled resources in the group.
`OFFLINE`	The resource group has been stopped on the node. In other words, the stopping methods (`Monitor_stop`, `Stop`, and `Postnet_stop`, as applicable to each resource) have executed successfully on all enabled resources in the group. This state also applies before a resource group has started for the first time on the node.
`PENDING_ONLINE`	The resource group is starting on the node. The starting methods (`Prenet_start`, `Start`, and `Monitor_start`, as applicable to each resource) are being executed on enabled resources in the group.
`PENDING_OFFLINE`	The resource group is stopping on the node. The stopping methods (`Monitor_stop`, `Stop`, and `Postnet_stop`, as applicable to each resource) are being executed on enabled resources in the group.
`ERROR_STOP_FAILED`	One or more resources within the resource group failed to stop successfully and are in `Stop_failed` state. Other resources in the group might remain

	online or offline. This resource group is not permitted to start on any node until the `ERROR_STOP_FAILED` state is cleared. You must use an administrative command, such as `scswitch -c`, to manually kill the `Stop_failed` resource and reset its state to `OFFLINE`.
`ONLINE_FAULTED`	The resource group was `PENDING_ONLINE` and has finished starting on this node. However, one or more resources ended up in `Start_failed` state or with `Faulted` status.
`PENDING_ONLINE_BLOCKED`	The resource group failed to start fully because one or more resources within that resource group have an unsatisfied strong resource dependency on a resource in a different resource group. Such resources remain `OFFLINE`. When the resource dependencies are satisfied, the resource group automatically moves back to `PENDING_ONLINE` state.

Default	No default
Tunable	Never

`RG_system` (`boolean`)

If the `RG_system` property is `TRUE` for a resource group, particular operations are restricted for the resource group and for the resources that the resource group contains. This restriction is intended to help prevent accidental modification or deletion of critical resource groups and resources. Only `scrgadm`(1M) and `scswitch`(1M) commands are affected by this property. Operations for `scha_control`(1HA) and `scha_control`(3HA) are not affected.

Before performing a restricted operation on a resource group (or a resource group's resources), you must first set the `RG_system` property of the resource group to `FALSE`. Use care when you modify or delete a resource group that supports cluster services, or when you modify or delete the resources that such a resource group contains.

The following table shows the operations that are restricted for a resource group when `RG_system` is set to `TRUE`.

Operation	Example
Delete a resource group	**scrgadm -r -g RG1**
Edit a resource group property (except for RG_system)	**scrgadm -c -t RG1 -y nodelist=...**
Add a resource to a resource group	**scrgadm -a -j R1 -g RG1**
Delete a resource from a resource group	**scrgadm -r -j R1 -g RG1**
Edit a property of a resource that belongs to a resource group	**scrgadm -c -j R1**
Switch a resource group offline	**scswitch -F -g RG1**
Manage a resource group	**scswitch -o -g RG1**
Unmanage a resource group	**scswitch -u -g RG1**
Enable a resource	**scswitch -e -j R1**
Enable monitoring for a resource	**scswitch -e -M -j R1**
Disable a resource	**scswitch -n -j R1**
Disable monitoring for a resource	**scswitch -n -M -j R1**

If the RG_system property is TRUE for a resource group, the only property of the resource group that you can edit is the RG_system property itself. In other words, editing the RG_system property is never restricted.

Default FALSE

Tunable Any time

SEE ALSO projects(1), scha_control(1HA), scrgadm(1M), scswitch(1M), scha_control(3HA), property_attributes(5), r_properties(5), rt_properties(5)

System Administration Guide: Resource Management and Network Services

NAME

r_properties – resource properties

DESCRIPTION

The list below describes the resource properties defined by Sun Cluster. These descriptions have been developed for data service developers. For more information about a particular data service, see that data service's man page.

Note – Scalable, as used in this man page, specifically describes a resource that uses the network load balancing features of Sun Cluster. Such a resource also uses the properties `Affinity_timeout`, `Load_balancing_policy`, `Load_balancing_weights`, `Port_list`, `UDP_affinity`, and `Weak_affinity`. Some resource types can run on multiple nodes without using network load balancing. The `Scalable` resource for such a resource is set to `False`, and such a resource does not use the preceding additional properties.

Resource Property Values

Required	The cluster administrator must specify a value when creating a resource with an administrative utility.
Optional	If the cluster administrator does not specify a value when creating a resource group, the system supplies a default value.
Conditional	The RGM creates the property only if the property is declared in the RTR file. Otherwise, the property does not exist and is not available to the cluster administrator. A conditional property declared in the RTR file is optional or required, depending on whether a default value is specified in the RTR file. For details, see the description of each conditional property.
Query-only	Cannot be set directly by an administrative tool.

All properties that are designated as tunable can be edited by the cluster administrator by using the command:

```
# scrgadm -c -j resource -y property=new value
```

Resource Properties and Descriptions

Note – Property names, such as `Affinity_timeout` and `Cheap_probe_interval`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify property names.

`Affinity_timeout` (`integer`)

Length of time, in seconds, during which connections from a given client IP address for any service in the resource are sent to the same server node. If you set this property to `-1`, all connections are sent to the same node. If you set this property to `0`, all open connections are sent to the same node. If you set this property to *n*, for *n* number of seconds after the last connection has closed, all new connections are sent to the same node as the last connection.

In all cases, if the server node leaves the cluster as a result of a failure, a new server node is selected.

This property is relevant only when `Load_balancing_policy` is either `Lb_sticky` or `Lb_sticky_wild`. In addition, `Weak_affinity` must be set to `False` (the default value).

This property is used only for scalable services.

Category	Conditional/Optional
Default	0
Tunable	`Anytime`

`Cheap_probe_interval` (`integer`)
The number of seconds between invocations of a quick fault probe of the resource. This property is only created by the RGM and available to the cluster administrator if this property is declared in the RTR file.

This property is optional if a default value is specified in the RTR file. If the `Tunable` attribute is not specified in the resource type file, the `Tunable` value for the property is `When_disabled`.

Category	Conditional
Default	See above
Tunable	When disabled

`Extension properties`
The developer declares the resource type properties in the RTR file. The RTR file defines the initial configuration of the data service at the time the cluster administrator registers the data service with Sun Cluster. For information about the individual attributes you can set for extension properties, see `property_attributes(5)`.

Category	Conditional
Default	No default
Tunable	Depends on the specific property

`Failover_mode` (`enum`)
Modifies the recovery actions that the RGM takes when a resource fails to start or to stop successfully, or when a resource monitor finds a resource to be unhealthy and consequently requests a restart or failover.

`NONE`, `SOFT`, or `HARD` (method failures)

These settings affect only failover behavior when a start or stop method (`Prenet_start`, `Start`, `Monitor_stop`, `Stop`, `Postnet_stop`) fails. Once the resource has started successfully, `NONE`, `SOFT`, and `HARD` have no effect on subsequent resource restart or giveover behavior that the resource monitor initiates with the `scha_control` command or the `scha_control()` function. See the `scha_control(1HA)` and the `scha_control(3HA)` man pages. `NONE` indicates

that the RGM is not to take any recovery action when one of the previously listed start or stop methods fails. SOFT or HARD indicates that if a Start or Prenet_start method fails, the RGM is to relocate the resource's group to a different node. For Start or Prenet_start failures, SOFT and HARD are the same.

For failure of a stop method (Monitor_stop, Stop, or Postnet_stop), SOFT is the same as NONE. If Failover_mode is set to HARD when one of these stop methods fails, the RGM reboots the node to force the resource group offline. The RGM might then attempt to start the group on another node.

RESTART_ONLY or LOG_ONLY

Unlike NONE, SOFT, and HARD, which affect failover behavior when a start or stop method fails, RESTART_ONLY and LOG_ONLY affect all failover behavior. Failover behavior includes monitor-initiated (scha_control) restarts of resources and resource groups, and giveovers that are initiated by the resource monitor (scha_control). RESTART_ONLY indicates that the monitor can run scha_control to restart a resource or a resource group. The RGM allows Retry_count restarts within Retry_interval. If Retry_count is exceeded, no further restarts are permitted. If Failover_mode is set to LOG_ONLY, no resource restarts or giveovers are permitted. Setting Failover_mode to LOG_ONLY is the same as setting Failover_mode to RESTART_ONLY with Retry_count set to zero.

RESTART_ONLY or LOG_ONLY (method failures)

If a Prenet_start, Start, Monitor_stop, Stop, or Postnet_stop method fails, RESTART_ONLY and LOG_ONLY are the same as NONE. That is, the node is neither failed over nor rebooted.

Effect of Failover_mode settings on a data service

The effect that each setting for Failover_mode has on a data service depends on whether the data service is monitored or unmonitored and whether it is based on the Data Services Development Library (DSDL).

- A data service is monitored if it implements a Monitor_start method and monitoring of the resource is enabled. The RGM starts a resource monitor by executing the Monitor_start method after starting the resource itself. The resource monitor probes the health of the resource. If the probes fail, the resource monitor might request a restart or a failover by calling the scha_control() function. For DSDL-based resources, probes might reveal partial failure (degradation) or a complete failure of the data service. Repeated partial failures accumulate to a complete failure.
- A data service is unmonitored if it does not provide a Monitor_start method or monitoring of the resource has been disabled.
- DSDL-based data services include those that are developed with Agent Builder, through the GDS, or by using the DSDL directly. Some data services, HA Oracle for example, were developed without using the DSDL.

NONE, SOFT, or HARD (probe failures)

If you set Failover_mode to NONE, SOFT, or HARD and the data service is a monitored DSDL-based service, and if the probe fails completely, the monitor calls the scha_control() function to request a restart of the resource. If probes continue to fail, the resource is restarted up to a maximum of Retry_count number of times within Retry_interval. If the probes fail again after the Retry_count number of restarts is reached, the monitor requests a failover of the resource's group to another node.

If you set Failover_mode to NONE, SOFT, or HARD and the data service is an unmonitored DSDL-based service, the only failure that is detected is the death of the resource's process tree. If the resource's process tree dies, the resource is restarted.

If the data service is a not a DSDL-based service, the restart or failover behavior depends on how the resource monitor is coded. For example, the Oracle resource monitor recovers by restarting the resource or the resource group, or by failing over the resource group.

RESTART_ONLY (probe failures)

If you set Failover_mode to RESTART_ONLY and the data service is a monitored DSDL-based service, and if the probe fails completely, the resource is restarted Retry_count times within Retry_interval. However, if Retry_count is exceeded, the resource monitor exits, sets the resource status to FAULTED, and generates the status message "Application faulted, but not restarted. Probe quitting." At this point, although monitoring is still enabled, the resource is effectively unmonitored until it is repaired and restarted by the cluster administrator.

If you set Failover_mode to RESTART_ONLY and the data service is an unmonitored DSDL-based service, and if the process tree dies, the resource is *not* restarted.

If a monitored data service is not DSDL-based, the recovery behavior depends on how the resource monitor is coded. If you set Failover_mode to RESTART_ONLY, the resource or resource group can be restarted by a call to the scha_control() function Retry_count times within Retry_interval. If the resource monitor exceeds Retry_count, the attempt to restart fails. If the monitor calls the scha_control() function to request a failover, that request fails as well.

LOG_ONLY (probe failures)

If you set Failover_mode to LOG_ONLY for any data service, all scha_control() requests either to restart the resource or resource group or to fail over the group are precluded. If the data service is DSDL-based, a message is logged when a probe completely fails, but the resource is not restarted. If a probe fails completely more than Retry_count times within Retry_interval, the resource monitor exits, sets the resource status to FAULTED, and generates the

status message “Application faulted, but not restarted. Probe quitting.” At this point, although monitoring is still enabled, the resource is effectively unmonitored until it is repaired and restarted by the cluster administrator.

If you set `Failover_mode` to `LOG_ONLY` and the data service is an unmonitored DSDL-based service, and if the process tree dies, a message is logged but the resource is not restarted.

If a monitored data service is not DSDL-based, the recovery behavior depends on how the resource monitor is coded. If you set `Failover_mode` to `LOG_ONLY`, all `scha_control()` requests either to restart the resource or resource group or to fail over the group fail.

Category	Optional
Default	`NONE`
Tunable	`Anytime`

`Load_balancing_policy` (`string`)
A string that defines the load-balancing policy in use. This property is used only for scalable services. The RGM automatically creates this property if the `Scalable` property is declared in the RTR file.

`Load_balancing_policy` can take the following values:

- `Lb_weighted` (the default). The load is distributed among various nodes according to the weights set in the `Load_balancing_weights` property.
- `Lb_sticky`. The set of ports is known at the time the application resources are configured. A given client (identified by the client’s IP address) of the scalable service is always sent to the same node of the cluster.
- `Lb_sticky_wild`. The port numbers are not known in advance but are dynamically assigned. A given client (identified by the client’s IP address) that connects to an IP address of a wildcard sticky service is always sent to the same cluster node regardless of the port number to which that IP address is coming.

Category	Conditional/Optional
Default	`Lb_weighted`
Tunable	At creation

`Load_balancing_weights` (`string_array`)
For scalable resources only. The RGM automatically creates this property if the `Scalable` property is declared in the RTR file. The format is *weight@node,weight@node...*, where *weight* is an integer that reflects the relative portion of load distributed to the specified *node*. The fraction of load distributed to a node is the weight for this node divided by the sum of all weights. For example, `1@1,3@2` specifies that node 1 receives 1/4 of the load and node 2 receives 3/4. The empty string (“”), the default, sets a uniform distribution. Any node that is not assigned an explicit weight receives a default weight of 1. You can specify weight 0 to assign no load to a node.

If the `Tunable` attribute is not specified in the resource type file, the `Tunable` value for the property is `Anytime`. Changing this property revises the distribution for new connections only.

Category	Conditional/Optional
Default	Null
Tunable	`Anytime`

method`_timeout` for each callback method (`integer`)
A time lapse, in seconds, after which the RGM concludes that an invocation of the method has failed.

Note – You cannot specify a maximum value for a method timeout (using the `Max` attribute). Likewise, you cannot specify a minimum value of zero (`Min=0`).

Category	Conditional/Optional
Default	3,600 (one hour) if the method itself is declared in the RTR file.
Tunable	`Anytime`

`Monitored_switch` (`enum`)
You cannot directly set this property. Rather, it is set to `Enabled` or `Disabled` by the RGM if the cluster administrator enables or disables the monitor with an administrative utility. If disabled, the `Monitor_start` method will not be called on the resource until monitoring is enabled again. If the resource does not have a monitor callback method, this property evaluates to `Disabled`.

Category	Query-only
Default	Enabled if the resource type has monitoring methods; disabled otherwise.
Tunable	See description

`Network_resources_used` (`string_array`)
A list of logical host name or shared address network resources used by the resource. For scalable services, this property refers to shared address resources that usually are configured in a separate resource group. For failover services, this property refers to logical host name or shared address resources that might exist in the same resource group or in a different group. The RGM automatically creates this property if the `Scalable` property is declared in the RTR file. If the `Scalable` property is not declared in the RTR file, `Network_resources_used` is unavailable unless it is explicitly declared in the RTR file.

If the `Tunable` attribute is not specified in the RTR file, the `Tunable` value for the property is `At_creation`.

Category	Conditional/Required
Default	No default
Tunable	At creation

`Num_resource_restarts` on each cluster node (`integer`)
You cannot directly set this property, which is set by the RGM to the number of `scha_control Resource_restart` or `Resource_is_restarted` calls that have been made for this resource on this node within the past *n* seconds, where *n* is the value of the `Retry_interval` property of the resource. The resource restart counter is reset to zero by the RGM whenever a `scha_control` giveover is executed by this resource, whether the giveover attempt succeeds or fails. `scha_control` is described in more detail in `scha_control`(1HA) or `scha_control`(3HA).

If a resource type does not declare the `Retry_interval` property, the `Num_resource_restarts` property is not available for resources of that type.

Category	Query-only
Default	No default
Tunable	See description

`Num_rg_restarts` on each cluster node (`integer`)
You cannot directly set this property, which is set by the RGM to the number of `scha_control Restart` calls that have been made by this resource for the resource group to which it belongs on this node within the past *n* seconds, where *n* is the value of the `Retry_interval` property of the resource. If a resource type does not declare the `Retry_interval` property, the `Num_rg_restarts` property is not available for resources of that type.

Category	Query-only
Default	No default
Tunable	See description

`On_off_switch` (`enum`)
You cannot directly set this property. Rather, it is set to `Enabled` or `Disabled` by the RGM if the cluster administrator enables or disables the resource with an administrative utility. If disabled, a resource has no callbacks invoked until it is enabled again.

Category	Query-only
Default	Disabled
Tunable	See description

`Port_list` (`string_array`)
A comma-separated list of port numbers on which the server is listening. Appended to each port number is a slash (`/`) followed by the protocol that is being used by that port, for example, `Port_list=80/tcp` or `Port_list=80/tcp6,40/udp6`.

Possible protocols that you can specify include `tcp`, for only TCP IPv4, `tcp6`, for both TCP IPv4 and TCP IPv6, `udp`, for only UDP IPv4, or `udp6`, for both UDP IPv4 and UDP IPv6.

If the `Scalable` property is declared in the RTR file, the RGM automatically creates `Port_list`. Otherwise, this property is unavailable unless it is explicitly declared in the RTR file.

Setting up this property for Apache is described in the *Sun Cluster Data Service for Apache Guide for Solaris OS*.

Category	Conditional/Required
Default	No default
Tunable	Anytime

`R_description` (`string`)
A brief description of the resource.

Category	Optional
Default	The empty string
Tunable	`Anytime`

`Resource_dependencies` (`string_array`)
A list of resources in the same 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.

If this resource is brought offline at the same time as those in the list, this resource stops before those in the list. However, if this resource remains online or fails to stop, a resource in the list that is in a different resource group stops anyway. Resources in the list cannot be disabled unless this resource is disabled first.

By default in a resource group, application resources have an implicit strong resource dependency on network address resources. `Implicit_network_dependencies` in `rg_properties(5)` contains more information.

Within a resource group, `Prenet_start` methods are run in dependency order before `Start` methods. `Postnet_stop` methods are run in dependency order after `Stop` methods. In different resource groups, the dependent resource waits for the depended-on resource to finish `Prenet_start` and `Start` before it runs `Prenet_start`. The depended-on resource waits for the dependent resource to finish `Stop` and `Postnet_stop` before it runs `Stop`.

Category	Optional
Default	The empty list

Tunable Anytime

`Resource_dependencies_restart` (`string_array`)
A list of resources in the same or in different groups upon which this resource has a restart 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_restart` 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 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.

If this resource is brought offline at the same time as those in the list, this resource stops before those in the list. However, if this resource remains online or fails to stop, a resource in the list that is in a different resource group stops anyway. Resources in the list cannot be disabled unless this resource is disabled first.

This property works just as `Resource_dependencies` does, except that, if any resource in the restart dependency list is restarted, this resource is restarted. The restart of this resource occurs after the resource in the list comes back online.

Within a resource group, `Prenet_start` methods are run in dependency order before `Start` methods. `Postnet_stop` methods are run in dependency order after `Stop` methods. In different resource groups, the dependent resource waits for the depended-on resource to finish `Prenet_start` and `Start` before it runs `Prenet_start`. The depended-on resource waits for the dependent resource to finish `Stop` and `Postnet_stop` before it runs `Stop`.

Category Optional

Default The empty list

Tunable Anytime

`Resource_dependencies_weak` (`string_array`)
A list of resources in the same or in different groups upon which this resource has a weak dependency. A weak dependency determines the order of method calls within the group. The RGM calls the `Start` methods of the resources in this list before the `Start` method of this resource. The RGM calls the `Stop` methods of this resource before the `Stop` methods of those in the list. The resource can still start if those in the list fail to start or remain offline.

If this resource and a resource in its `Resource_dependencies_weak` list start concurrently, the RGM waits until the resource in the list starts before the RGM starts this resource. If the resource in the list does not start (for example, if the resource group for the resource in the list remains offline or the resource in the list is in a `Start_failed` state), this resource starts. This resource's resource group

might enter a `Pending_online_blocked` state temporarily as resources in the this resource's `Resource_dependencies_weak` list start. When all resources in the list have started or failed to start, this resource starts and its group re-enters the `Pending_online` state.

If this resource is brought offline at the same time as those in the list, this resource stops before those in the list. However, if this resource remains online or fails to stop, a resource in the list that is in a different resource group stops anyway. Resources in the list cannot be disabled unless this resource is disabled first.

Within a resource group, `Prenet_start` methods are run in dependency order before `Start` methods. `Postnet_stop` methods are run in dependency order after `Stop` methods. In different resource groups, the dependent resource waits for the depended-on resource to finish `Prenet_start` and `Start` before it runs `Prenet_start`. The depended-on resource waits for the dependent resource to finish `Stop` and `Postnet_stop` before it runs `Stop`.

Category	Optional
Default	The empty list
Tunable	`Anytime`

`Resource_name` (`string`)
The name of the resource instance. Must be unique within the cluster configuration and cannot be changed after a resource has been created.

Category	Required
Default	No default
Tunable	Never

`Resource_project_name` (`string`)
The Solaris project name (see `projects`(1)) associated with the resource. Use this property to apply Solaris resource management features such as CPU shares and resource pools to cluster data services. When the RGM brings resources online, it launches the related processes under this project name. If this property is not specified, the project name will be taken from the `RG_project_name` property of the resource group that contains the resource (see `rg_properties(5)`). If neither property is specified, the RGM uses the predefined project name "`default`". The specified project name must exist in the projects database(see `projects`(1) and *System Administration Guide: Resource Management and Network Services*).

This property is only supported starting in Solaris 9.

Note – Changes to this property take affect the next time the resource is started.

Category	Optional
Default	Null
Tunable	`Anytime`
Valid value	Any valid Solaris project name, or Null

`Resource_state` on each cluster node (`enum`)
The RGM-determined state of the resource on each cluster node. Possible states include: `Online`, `Offline`, `Start_failed`, `Stop_failed`, `Monitor_failed`, `Online_not_monitored`, `Starting`, and `Stopping`.

`Online`	The starting methods (`Prenet_start`, `Start`, and `Monitor_start`) have executed successfully on the resource on this node.
`Offline`	The resource has not yet started for the first time on this node, or the stopping methods (`Monitor_stop`, `Stop`, and `Postnet_stop`, as applicable to the particular resource) have executed successfully on the resource on this node.
`Start_failed`	A `Prenet_start` or `Start` method failed on the resource on this node. `Failed` means that the method exited with a nonzero exit status or timed out. The service that is represented by the resource might or might not actually have started on this node.
`Stop_failed`	A `Monitor_stop`, `Stop`, or `Postnet_stop` method failed on the resource on this node. `Failed` means that the method exited with a nonzero exit status or timed out. The service that is represented by the resource might or might not actually have stopped on this node. When a resource enters this state, the resource group state becomes `Error_stop_failed` and requires you to intervene. `Error_stop_failed` is described in more detail in `rg_properties(5)`.
`Monitor_failed`	The resource successfully executed its `Prenet_start` or `Start` methods (as applicable to the specific resource type). However, the resources' `Monitor_start` method exited with a nonzero exit status or timed out. The resource monitor might or might not actually have started on this node.

`Online_not_monitored`	The resource successfully executed its `Prenet_start` or `Start` methods (as applicable to the specific resource type). The `Monitor_start` method has not yet been executed on the resource. A resource that is unmonitored (that is, for which there is no `Monitor_start` method, or for which monitoring has been disabled) remains in this state when the resource group goes to `Online` state.
`Starting`	The resource is running the `Prenet_start` or `Start` method in an attempt to go online.
`Stopping`	The resource is running the `Start` or `Postnet_stop` method in an attempt to go offline.

You cannot configure this property.

Category	Query-only
Default	No default
Tunable	Never

`Retry_count` (`integer`)

The number of times a monitor attempts to restart a resource if it fails. If the `Retry_count` is exceeded, depending on the particular data service and the setting of the `Failover_mode` property, the monitor might do one of the following:

- Allow the resource group to remain on the current primary, even though the resource is in a faulted state.
- Request a failover of the resource group onto a different node.

This property is created by the RGM and is made available to the cluster administrator only if this property is declared in the RTR file. This property is optional if a default value is specified in the RTR file.

If the `Tunable` attribute is not specified in the resource type file, the `Tunable` value for the property is `When_disabled`.

Note – If you specify a negative value for this property, the monitor attempts to restart the resource an unlimited number of times.

Category	Conditional
Default	See above
Tunable	When disabled

`Retry_interval` (`integer`)
The number of seconds in which to count attempts to restart a failed resource. The resource monitor uses this property in conjunction with `Retry_count`. This property is created by the RGM and made available to the cluster administrator only if it is declared in the RTR file. This property is optional if a default value is specified in the RTR file.

If the `Tunable` attribute is not specified in the resource type file, the `Tunable` value for the property is `When_disabled`.

Note – If the `Retry_interval` property is not declared, the call to `scha_resource_get (num_*_restarts)` fails with exit 13 (`SCHA_ERR_RT`).

Category	Conditional
Default	See above
Tunable	When disabled

`Scalable` (`boolean`)
Indicates whether the resource is scalable, that is, whether the resource uses the networking load balancing features of Sun Cluster.

If this property is declared in the RTR file, the RGM automatically creates the following scalable service properties for resources of that type: `Affinity_timeout`, `Load_balancing_policy`, `Load_balancing_weights`, `Network_resources_used`, `Port_list`, `UDP_affinity`, and `Weak_affinity`. These properties have their default values unless they are explicitly declared in the RTR file. The default for `Scalable`, when it is declared in the RTR file, is `True`.

If this property is declared in the RTR file, it is not permitted to be assigned a `Tunable` attribute other than `At_creation`.

If this property is not declared in the RTR file, the resource is not scalable, you cannot tune this property, and no scalable service properties are set by the RGM. However, you can explicitly declare the `Network_resources_used` and `Port_list` properties in the RTR file, if you want, because these properties can be useful in a non-scalable service as well as in a scalable service.

You use this resource property in combination with the `Failover` resource type property, as follows:

If `Failover` is	If `Scalable` is	Description
`True`	`True`	Do not specify this illogical combination.
`True`	`False`	Specify this combination for a failover service.

If `Failover` is	If `Scalable` is	Description
`False`	`True`	Specify this combination for a scalable service that uses a `SharedAddress` resource for network load balancing. The *Sun Cluster Concepts Guide* describes `SharedAddress` in more detail.
`False`	`False`	Although it is an unusual combination, you can use this combination to configure a multi-master service that does not use network load balancing.

The description for `Failover` in `rt_properties(5)` contains additional information.

Category Optional

Default See above

Tunable At creation

`Status` on each cluster node (`enum`)
Set by the resource monitor. Possible values are: `Online`, `Degraded`, `Faulted`, `Unknown`, and `Offline`. The RGM sets the value to `Online` when the resource is started, if it is not already set by the `Start` (or `Prenet_start`) method, to `Offline` when the resource is stopped, if it is not already set by the `Stop` (or `Postnet_stop`) method.

Category Query-only

Default No default

Tunable Only by using `scha_resource_setstatus(1HA)`

`Status_msg` on each cluster node (`string`)
Set by the resource monitor at the same time as the `Status` property. The RGM sets it to the empty string when the resource is brought `Offline`, if it was not already set by the `Stop` (or `Postnet_stop`) method.

Category Query-only

Default No default

Tunable Only by using `scha_resource_setstatus(1HA)`

`Thorough_probe_interval` (`integer`)
The number of seconds between invocations of a high-overhead fault probe of the resource. This property is created by the RGM and available to the cluster administrator only if it is declared in the RTR file. This property is optional if a default value is specified in the RTR file.

If the `Tunable` attribute is not specified in the resource type file, the `Tunable` value for the property is `When_disabled`.

Category	Conditional
Default	No default
Tunable	When disabled

`Type` (`string`)
An instance's resource type.

Category	Required
Default	No default
Tunable	Never

`Type_version` (`string`)
Specifies which version of the resource type is currently associated with this resource. The RGM automatically creates this property, which cannot be declared in the RTR file. The value of this property is equal to the `RT_version` property of the resource's type. When a resource is created, the `Type_version` property is not specified explictly, though it may appear as a suffix of the resource type name. When a resource is edited, the `Type_version` may be changed to a new value.

Category	See above
Default	None
Tunable	Its tunability is derived from: ■ The current version of the resource type ■ The `#$upgrade_from` directive in the resource type registration file (see `rt_reg`(4))

`UDP_affinity` (`boolean`)
If true, all UDP traffic from a given client is sent to the same server node that currently handles all TCP traffic for the client.

This property is relevant only when `Load_balancing_policy` is either `Lb_sticky` or `Lb_sticky_wild`. In addition, `Weak_affinity` must be set to False (the default value).

This property is only used for scalable services.

Category	Conditional/Optional
Default	False
Tunable	When disabled

`Weak_affinity` (`boolean`)
If true, enable the weak form of the client affinity. This allows connections from a given client to be sent to the same server node except when a server listener starts (for example, due to a fault monitor restart, a resource failover or switchover, or a node rejoining a cluster after failing) or when `load_balancing_weights` for the scalable resource changes due to an administration action.

Weak affinity provides a low overhead alternative to the default form, both in terms of memory consumption and processor cycles.

This property is relevant only when `Load_balancing_policy` is either `Lb_sticky` or `Lb_sticky_wild`.

This property is only used for scalable services.

Category	Conditional/Optional
Default	False
Tunable	When disabled

SEE ALSO `projects`(1), `scha_control(1HA)`, `scha_resource_setstatus(1HA)`, `scrgadm(1M)`, `scha_control(3HA)`, `rt_reg(4)`, `property_attributes(5)`, `rg_properties(5)`, `rt_properties(5)`

Sun Cluster Data Services Developer's Guide for Solaris OS, *System Administration Guide: Resource Management and Network Services*

NAME

rt_properties – resource type properties

DESCRIPTION

The following information describes the resource type properties that are defined by Sun Cluster. These descriptions have been developed for data service developers. For information about a particular data service, see that data service man page.

Resource Type Property Values

Required	The property requires an explicit value in the Resource Type Registration (RTR) file. Otherwise, the object to which the property belongs cannot be created. A blank or the empty string is not allowed as a value.
Conditional	To exist, the property must be declared in the RTR file. Otherwise, the RGM does not create the property and the property is not available to administrative utilities. A blank or the empty string is allowed. If the property is declared in the RTR file but no value is specified, the RGM supplies a default value.
Conditional/Explicit	To exist, the property must be declared in the RTR file with an explicit value. Otherwise, the RGM does not create the property and the property is not available to administrative utilities. A blank or the empty string is not allowed.
Optional	The property can be declared in the RTR file. If the property is not declared in the RTR file, the RGM creates it and supplies a default value. If the property is declared in the RTR file but no value is specified, the RGM supplies the same default value as if the property were not declared in the RTR file.
Query-only	The property cannot be set directly by an administrative utility. These properties are not set in the RTR file.

Note – Resource type properties cannot be updated by administrative utilities with the exception of `Installed_nodes` and `RT_system`, which cannot be declared in the RTR file and must be set by the administrator.

Resource Type Properties and Descriptions

A resource type is defined by a resource type registration file that specifies standard and extension property values for the resource type.

Note – Resource type property names, such as `API_version` and `Boot`, are *not* case sensitive. You can use any combination of uppercase and lowercase letters when you specify property names.

`API_version` (`integer`)
The version of the resource management API that is used by this resource type implementation.

The following information summarizes the maximum `API_version` that is supported by each release of Sun Cluster.

Before and up to 3.1	2
3.1 10/03	3
3.1 4/04	4
3.1 9/04	5
3.1 3/05	6

Declaring a value for `API_version` that is greater than 2 in the RTR file prevents that resource type from being installed on a version of Sun Cluster that supports a lower maximum version. For example, if you declare `API_version=5` for a resource type, that resource type cannot be installed on any version of Sun Cluster that was released before 3.1 9/04.

Category	Optional
Default	2
Tunable	Never

`Boot` (`string`)
An optional callback method: the path to the program that the RGM invokes on a node, which joins or rejoins the cluster when a resource of this type is already managed. This method is expected to initialize resources of this type similar to the `Init` method.

Category	Conditional/Explicit
Default	None
Tunable	Never

`Failover` (`boolean`)
`TRUE` indicates that resources of this type cannot be configured in any group that can be online on multiple nodes at once.

You use this resource type property in combination with the `Scalable` resource property, as follows:

If `FAILOVER` is	If `Scalable` is	Description
`TRUE`	`TRUE`	Do not specify this illogical combination.
`TRUE`	`FALSE`	Specify this combination for a failover service.

If FAILOVER is	If Scalable is	Description
FALSE	TRUE	Specify this combination for a scalable service that uses a SharedAddress resource for network load balancing. The *Sun Cluster Concepts Guide* describes SharedAddress in more detail.
FALSE	FALSE	Although it is an unusual combination, you can use this combination to select a multi-master service that does not use network load balancing.

The description for Scalable in r_properties(5) and "Key Concepts – Administration and Application Development" in *Sun Cluster Concepts Guide for Solaris OS* contain additional information.

Category Optional

Default FALSE

Tunable Never

Fini (string)
An optional callback method: the path to the program that the RGM invokes when a resource of this type is removed from RGM management.

Category Conditional/Explicit

Default No default

Tunable Never

Init (string)
An optional callback method: the path to the program that the RGM invokes when a resource of this type becomes managed by the RGM.

Category Conditional/Explicit

Default No default

Tunable Never

Init_nodes (enum)
The values can be RG_primaries (just the nodes that can master the resource) or RT_installed_nodes (all nodes on which the resource type is installed). Indicates the nodes on which the RGM is to call the Init, Fini, Boot, and Validate methods.

Category Optional

Default RG_primaries

Tunable Never

`Installed_nodes` (`string_array`)
A list of the cluster node names that the resource type is allowed to be run on. The RGM automatically creates this property. The cluster administrator can set the value. You cannot declare this property in the RTR file.

Category	Can be configured by the cluster administrator
Default	All cluster nodes
Tunable	Any time

`Is_logical_hostname` (`boolean`)
`TRUE` indicates that this resource type is some version of the `LogicalHostname` resource type that manages failover Internet Protocol (IP) addresses.

Category	Query-only
Default	No default
Tunable	Never

`Is_shared_address` (`boolean`)
`TRUE` indicates that this resource type is some version of the `SharedAddress` resource type that manages failover IP (Internet Protocol) addresses.

Category	Query-only
Default	No default
Tunable	Never

`Monitor_check` (`string`)
An optional callback method: the path to the program that the RGM invokes before doing a monitor-requested failover of a resource of this type.

Category	Conditional/Explicit
Default	No default
Tunable	Never

`Monitor_start` (`string`)
An optional callback method: the path to the program that the RGM invokes to start a fault monitor for a resource of this type.

Category	Conditional/Explicit
Default	No default
Tunable	Never

`Monitor_stop` (`string`)
A callback method that is required if `Monitor_start` is set: the path to the program that the RGM invokes to stop a fault monitor for a resource of this type.

Category	Conditional/Explicit
Default	No default

Tunable Never

`Pkglist` (`string_array`)
An optional list of packages that are included in the resource type installation.

Category Conditional/Explicit

Default No default

Tunable Never

`Postnet_stop` (`string`)
An optional callback method: the path to the program that the RGM invokes after calling the `Stop` method of any network-address resources on which a resource of this type depends. After the network interfaces are configured down, this method must perform `Stop` actions.

Category Conditional/Explicit

Default No default

Tunable Never

`Prenet_start` (`string`)
An optional callback method: the path to the program that the RGM invokes before calling the `Start` method of any network-address resources on which a resource of this type depends. This method is expected to perform `Start` actions that must be performed before network interfaces are configured.

Category Conditional/Explicit

Default No default

Tunable Never

`Resource_list` (`string_array`)
The list of all resources of the resource type. The administrator does not set this property directly. Rather, the RGM updates this property when the administrator adds or removes a resource of this type to or from any resource group.

Category Query-only

Default Empty list

Tunable Never

`Resource_type` (`string`)
The name of the resource type. To view the names of the currently registered resource types, use:

`scrgadm -p`

In Sun Cluster 3.1 and later releases, a resource type name includes the version, which is mandatory:

`vendor_id.resource_type:version`

The three components of the resource type name are properties that are specified in the RTR file as *Vendor_id*, *Resource_type*, and *RT_version*. The `scrgadm` command inserts the period (.) and colon (:) delimiters. The `RT_version` suffix of the resource type name is the same value as the `RT_version` property. To ensure that the *Vendor_id* is unique, the recommended approach is to use the stock symbol for the company creating the resource type. Resource type names that were created before Sun Cluster 3.1 continue to use the syntax:

`vendor_id.resource_type`

Category	Required
Default	Empty string
Tunable	Never

`RT_basedir` (`string`)
The directory path that is used to complete relative paths for callback methods. This path is expected to be set to the installation location for the resource type packages. The path must be a complete path, that is, the path must start with a forward slash (`/`). This property is not required if all the method path names are absolute.

Category	Required unless all method path names are absolute
Default	No default
Tunable	Never

`RT_description` (`string`)
A brief description of the resource type.

Category	Conditional
Default	Empty string
Tunable	Never

`RT_system` (`boolean`)
If the `RT_system` property is `TRUE` for a resource type, you cannot delete the resource type (`scrgadm -r -t` *resource_type_name*). This property is intended to help prevent accidental deletion of resource types, such as `LogicalHostname`, that are used to support the cluster infrastructure. However, you can apply the `RT_system` property to any resource type.

To delete a resource type whose `RT_system` property is set to `TRUE`, you must first set the property to `FALSE`. Use care when you delete a resource type whose resources support cluster services.

Category	Optional
Default	`FALSE`
Tunable	Any time

`RT_version` (`string`)
Starting in Sun Cluster 3.1, a required version string of this resource type implementation. The `RT_version` is the suffix component of the full resource type name. The `RT_version` property, which was optional in Sun Cluster 3.0, is mandatory in Sun Cluster 3.1 and later releases.

Category	Conditional/Explicit or Required
Default	No default
Tunable	Never

`Single_instance` (`boolean`)
If `TRUE`, indicates that only one resource of this type can exist in the cluster. The RGM allows only one resource of this type to run cluster-wide at one time.

Category	Optional
Default	`FALSE`
Tunable	Never

`Start` (`string`)
A callback method: the path to the program that the RGM invokes to start a resource of this type.

Category	Required unless the RTR file declares a `Prenet_start` method
Default	No default
Tunable	Never

`Stop` (`string`)
A callback method: the path to the program that the RGM invokes to stop a resource of this type.

Category	Required unless the RTR file declares a `Postnet_stop` method
Default	No default
Tunable	Never

`Update` (`string`)
An optional callback method: the path to the program that the RGM invokes when properties of a running resource of this type are changed.

Category	Conditional/Explicit
Default	No default
Tunable	Never

`Validate` (`string`)
An optional callback method: the path to the program that will be invoked to check values for properties of resources of this type.

Category	Conditional/Explicit

Default	No default
Tunable	Never

`Vendor_ID` (`string`)
See the `Resource_type` property.

Category	Conditional
Default	No default
Tunable	Never

SEE ALSO `scrgadm`(1M), `rt_reg`(4), `property_attributes`(5), `r_properties`(5), `rg_properties`(5)

NAME

scalable_service – scalable resource types

DESCRIPTION

A scalable data service is one that takes advantage of the Sun Cluster networking facility. Such a service is implemented as a resource type managed by the Resource Group Manager (RGM).

Standard Resource Properties

The standard resource properties `Scalable`, `Network_resources_used`, `Port_list`, `Load_balancing_policy`, and `Load_balancing_weights` are common to all scalable resource types. See `scrgadm(1M)` for the syntax and description of these properties.

Some data services can run in either a scalable or non-scalable mode. Such services permit you to specify a value of `True` or `False` for the `Scalable` property at the time the resource is created. If this property is set to `True` on a resource, the resource is said to be in "scalable mode." The resource then must be contained in a scalable mode resource group, that is, a group that can have its `Maximum_primaries` property set greater than 1.

For a data service that can only run in scalable mode, the `Scalable` property is implicitly True for resources of this type, and cannot be changed by the adminstrator.

You can change the `Load_balancing_weights` and `Port_list` properties at any time, even while the resource is online. `Network_resources_used` and `Load_balancing_policy` are set when the resource is created, and you cannot edit these properties afterward. Depending on how the resource type is implemented, these properties might have default values, or you might be required to provide values at when you create the resource.

Network Monitoring

A scalable service instance running on a particular node needs to be able to reply to clients over the public networks. The RGM automatically monitors the health of the public networks on nodes where scalable services are to run, and might bring down a scalable service instance on a particular node if the public network becomes inaccessible from that node. If monitoring is disabled on a scalable resource using `scswitch -n -M -j`, these network checks are disabled.

Resource Validatation

When the `Scalable` resource property that is set to `True` is created or updated, the RGM validates various resource properties and will reject the attempted update if these properties are not configured correctly. Among the checks that are performed are the following:

- The `Network_resources_used` property must not be empty. It must contain the names of existing `SharedAddress` resources. Every node in the `Nodelist` of the resource group containing the scalable resource must appear in either the `NetIfList` property or the `AuxNodeList` property of one of the named `SharedAddress` resources.
- The resource group that contains the scalable resource must have its `RG_dependencies` property set to include the resource groups of all `SharedAddress` resources listed in the scalable resource's `Network_resources_used` property.

- The `Port_list` property must not be empty. It must contain a list of port and protocol pairs, where protocol is `tcp`, `tcp6`, `udp`, or `udp6`. Possible protocols that you can specify include `tcp` for only TCP IPv4, `tcp6` for both TCP IPv4 and TCP IPv6, `udp` for only UDP IPv4, or `udp6` for both UDP IPv4 and UDP IPv6.

  For example, you can specify `Port_list=80/tcp,40/udp`.

Affinity

IP affinity guarantees that connections from a given client IP address are forwarded to the same cluster node. `Affinity_timeout`, `UDP_affinity`, and `Weak_affinity` are only relevant when `Load_balancing_policy` is set to either `Lb_sticky` or `Lb_sticky_wild`. See `r_properties(5)` for detail information.

SEE ALSO

`rt_callbacks(1HA)`, `scrgadm(1M)`, `rt_reg(4)`, `r_properties(5)`

Sun Cluster Software Installation Guide for Solaris OS, *Sun Cluster Data Services Developer's Guide for Solaris OS*

NAME

SUNW.Event – resource type implementation for the Cluster Reconfiguration Notification Protocol (CRNP)

DESCRIPTION

The `SUNW.Event` resource type implementation provides highly available CRNP services on Sun Cluster. This implementation makes the notification daemon (`/usr/cluster/lib/sc/cl_apid`) highly available by managing it as a resource under the Sun Cluster resource group manager (RGM). The resource group that contains the `SUNW.Event` resource must have a network resource configured in the same resource group. Only a single resource of type `SUNW.Event` should exist on a cluster.

Standard Properties

This section describes key standard properties that control the behavior of the implementation. You use `scrgadm(1M)` to set these properties on a `SUNW.Event` resource. `r_properties(5)` describes these resource properties in more detail.

`Network_resources_used` (`string_array`)

A comma-separated list of logical host name or shared address network resources that are used by the resource. `r_properties(5)` describes `Network_resources_used` in more detail.

Category	Conditional/Required
Default	No default
Tunable	When disabled

`Port_list` (`string_array`)

A comma-separated list of port numbers on which the server is listening. The `r_properties(5)` man page describes `Port_list` in more detail.

Category	Conditional/Required
Default	No default
Tunable	Anytime

`Retry_count` (`integer`)

The number of times that a monitor attempts to restart a resource if it fails. The `r_properties(5)` man page describes `Retry_count` in more detail.

Note – If you specify a negative value for this property, the monitor attempts to restart the resource an unlimited number of times.

Category	Conditional
Default	2
Tunable	Anytime

`Retry_interval` (`integer`)
: The number of seconds over which to count attempts to restart a failed resource. `r_properties(5)` describes `Retry_interval` in more detail.

Category	Conditional
Default	300
Tunable	Anytime

`Thorough_probe_interval` (`integer`)
: The number of seconds between invocations of a high overhead fault probe of the resource. `r_properties(5)` describes `Thorough_probe_interval` in more detail.

Category	Conditional
Default	60
Tunable	Anytime

Extension Properties

This section describes key extension properties that control the behavior of the implementation.

`Allow_hosts` (`string_array`)
: This property controls the set of clients that are allowed to register with the implementation to recieve cluster reconfiguration events. The general form of this property is `ipaddress/masklength`, which defines a subnet from which the clients are allowed to register. For example, the setting 129.99.77.0/24 allows clients on the subnet 129.99.77 to register for events. As another example, 192.9.84.231/32 allows only the client 192.9.84.231 to register for events.

  In addition, the following special keywords are recognized. `LOCAL` refers to all clients that are located in directly connected subnets of the cluster. `ALL` allows all clients to register. Note that if a client matches an entry in both the `Allow_hosts` and the `Deny_hosts` property, that client is prevented from registering with the implementation.

Category	Optional
Default	`LOCAL`
Tunable	Anytime

`Client_retry_count` (`integer`)
: This property controls the number of attempts made by the implementation while communicating with external clients. If a client fails to respond within

	`Client_retry_count` attempts, the client times out. The client is subsequently removed from the list of registered clients that are eligible to recieve cluster reconfiguration events. The client must re-register in order to start recieving events again. The section about the `Client_retry_interval` property describes how often these retries are made by the implementation. Category Optional Default 3 Tunable Anytime
`Client_retry_interval` (`integer`)	This property defines the time period (in seconds) used by the implementation while communicating with unresponsive external clients. Up to `Client_retry_count` attempts are made during this interval to contact the client. The value for this property can be modified at any time. Category Optional Default 1800 Tunable Anytime
`Client_timeout` (`integer`)	This property is the time out value (in seconds) that is used by the implementation while communicating with external clients. However, the implementation continues to attempt to contact the client for a tunable number of times. The sections about the `Client_retry_count` and `Client_retry_interval` properties describe the means of tuning this property. Category Optional Default 60 Tunable Anytime
`Deny_hosts` (`string_array`)	This property controls the set of clients that are prevented from registering to recieve cluster reconfiguration events. To determine access, the settings on this property take precedence over those in the `Allow_hosts` list. The format of this property is the same as the format that is defined in the `Allow_hosts`.

	Category	Optional
	Default	NULL
	Tunable	Anytime

`Max_clients` (integer) — This property controls the maximum number of clients that can register with the implementation to recieve notification of cluster events. Attempts by additional clients to register for events are rejected by the implementation. Since each client registration uses resources on the cluster, tuning this property allows users to control resource usage on the cluster by external clients.

Category	Optional
Default	1000
Tunable	Anytime

EXAMPLES

EXAMPLE 1 Creating a `SUNW.Event` Resource With Default Properties

This example shows how to create a failover `SUNW.Event` resource that is named `CRNP` in an existing resource group that is named `events-rg`. `events-rg` contains a `LogicalHostname` or `SharedAddress` resource, which identifies the failover host name that is associated with the resource group.

```
# scrgadm -a -t SUNW.Event
# scrgadm -a -j CRNP -t SUNW.Event -g events-rg
```

In this example, the `SUNW.Event` resource that is created is named `CRNP`. This resource listens on port 9444 and allows all clients on directly connected subnets to register for events.

EXAMPLE 2 Creating a `SUNW.Event` Resource With Non-Default Properties

This example shows how to create a `SUNW.Event` resource that is named `CRNP` in a resource group that is named `events-rg`. The `CRNP` resource is configured to listen on port 7000, and a specific network resource `foo-1` (already configured in the `events-rg`). This CRNP resource allows clients on subnet 192.9.77.0 and clients on directly connected subnets to register, but disallows the client 192.9.77.98 from using the implementation.

```
# scrgadm -a -g events-rg -j CRNP -t SUNW.Event -y \
Port_list=7000/tcp -y Network_resources_used=foo-1 -x \
Allow_hosts=LOCAL,192.9.77.0/24 -x Deny_hosts=192.9.77.98/32
```

FILES

`/usr/cluster/lib/sc/cl_apid`
CRNP daemon

`/usr/cluster/lib/sc/events/dtds`
Directory that contains data type definitions for the CRNP protocol

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes.

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu

SEE ALSO `scrgadm`(1M), `scswitch`(1M), `scha_resource_get`(1HA), `attributes`(5), `r_properties`(5)

NAME

SUNW.gds – resource type for making simple network aware and non-network aware applications highly available or scalable

DESCRIPTION

The Generic Data Service (GDS) is a mechanism that enables you to make simple network-aware and non-network aware applications highly available or scalable by plugging them into the Sun Cluster Resource Group Manager (RGM) framework.

The GDS contains a fully functional Sun Cluster resource type, complete with callback methods (`rt_callbacks(1HA)`) and a Resource Type Registration (RTR) file (`rt_reg(4)`).

Standard Properties

`Network_resources_used`
For a network-aware application, if this property is omitted, the application needs to listen on all addresses. This property need not be specified unless the application binds to one or more specific addresses. `r_properties(5)` contains more detail.

Before creating the network-aware, GDS resource, a `LogicalHostname` or `SharedAddress` resource must already have been configured in the same resource group as the GDS resource.

Category	Optional
Default	Null
Tunable	When disabled

`Port_list`
A comma-separated list of port numbers on which the server is listening. The `r_properties(5)` man page describes `Port_list` in more detail.

Category	Required (only if the application is network-aware)
Default	No default
Tunable	Anytime

`Start_timeout` (`integer`)
This property specifies the timeout value, in seconds, for the start command.

Category	Optional
Default	300 seconds
Tunable	Any time

`Stop_timeout` (`integer`)
This property specifies the timeout value, in seconds, for the stop command.

Category	Optional
Default	300 seconds
Tunable	Any time

Extension Properties

`Start_command` (`string`)
The `start` command starts the application. This command must be a complete command line that can be passed directly to a shell to start the application.

Category	Required
Default	No default
Tunable	When disabled

`Stop_command` (`string`)
The `stop` command for the application. This command must be a complete command line that can be passed directly to a shell to stop the application. If this property is omitted, the GDS stops the application by using signals.

Category	Optional
Default	Null
Tunable	When disabled

`Probe_command` (`string`)
The `probe` command periodically checks the health of a network aware or non-network aware application. It must be a complete command line 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 failure of the application. This exit status, called probe status, is an integer between `0` (for success) and `100` (for complete failure). The probe status can also be `201`, which causes the application to fail over unless `Failover_enabled` is set to `False`.

The probe status is used within the GDS probing algorithm to decide whether to restart the application locally or to fail over the application to another node. If the `probe` command is omitted, the GDS provides its own simple probe that connects to the application on the network resource. If the connect succeeds, the GDS disconnects immediately. If both connect and disconnect succeed, the application is deemed to be running correctly.

The GDS does not provide "default" probing behavior for non-network aware applications. However, a non-network aware application is started under the Process Monitor Facility (PMF), which monitors the application and restarts the application if it fails to remain alive. The `pmfadm`(1M) man page contains more information.

Category	Optional
Default	Null
Tunable	When disabled

`Probe_timeout` (`integer`)
This property specifies the timeout value, in seconds, for the probe command.

Category	Optional
Default	30 seconds

Tunable Any time

`Child_mon_level` (`integer`)
This property provides control over the processes that are monitored through the Process Monitor Facility (PMF). This property denotes the level to which the forked children processes are monitored. Omitting this property or setting this property to the default value is the same as omitting the `-C` option for `pmfadm`(1M): all children (and their descendents) are monitored.

Category Optional

Default `-1`

Tunable At creation

`Failover_enabled` (`boolean`)
This property allows the resource to fail over. If this property is set to `False`, failover of the resource is disabled. You can use this property to prevent the application resource from initiating a failover of the resource group.

Category Optional

Default `True`

Tunable When disabled

`Stop_signal` (`integer`)
This property specifies the signal that is to stop the application. The values of this property are the same as those defined in `signal`(3HEAD).

Category Optional

Default `15`

Tunable When disabled

`Log_level` (`enum`)
This property specifies the level, or type, of diagnostic messages that are logged by GDS. You can specify `None`, `Info`, or `Err` for this property. When you specify `None`, diagnostic messages are not logged by GDS. When you specify `Info`, both information and error messages are logged. When you specify `Err`, only error messages are logged.

Category Optional

Default `Info`

Tunable Any time

`Network_aware` (`boolean`)
This property specifies whether an application uses the network.

Category Optional

Default `True`

Tunable At creation

EXAMPLES

The following examples show how to use GDS to make an application named `app` highly available. You can also use SunPlex Agent Builder (`scdsbuilder`(1HA)) to create scripts that contain these commands.

Basic Example

This example shows how to register the `SUNW.gds` resource type, create a resource group for the application, create the `LogicalHostname` resource for the logical host name `hhead`, create the application resource, and then use `scswitch`(1M) to manage the resource group, enable all the resources, and bring the resources online.

At this point, the application is up and running in a highly available fashion and is being monitored by the simple probe that is provided by GDS. You can now use `scstat`(1M) to check the status of the application.

```
# scrgadm -a -t SUNW.gds
# scrgadm -a -g rg1
# scrgadm -a -L -g rg1 -l hhead
# scrgadm -a -t SUNW.gds -g rg1 -j app-rs \
          -x Start_command="/usr/local/app/bin/start" \
          -y Port_list="1234/tcp"
# scswitch -Z -g rg1
# scstat -g
```

Complex Example

This example shows how to register the `SUNW.gds` resource type, create a resource group for the application, create the `LogicalHostname` resource for the logical host name `hhead`, create the application resource, log error messages only, and then use `scswitch` to manage the resource group, enable all the resources, and bring the resources online.

At this point, the application is up and running in a highly available fashion and is being monitored by the fault monitor that is specified by `Probe_command`. You can now use `scstat` to check the status of the application.

```
# scrgadm -a -t SUNW.gds
# scrgadm -a -g rg1
# scrgadm -a -L -g rg1 -l hhead
# scrgadm -a -t SUNW.gds -g rg1 -j app-rs \
          -x Start_command="/usr/local/app/bin/start" \
          -x Stop_command="/usr/local/app/bin/stop"   \
          -x Probe_command="/usr/local/app/bin/probe" \
          -x stop_signal=9 -x failover_enabled=false  \
          -y Start_timeout=120 -y Stop_timeout=180    \
          -y Port_list="1234/tcp" -x Probe_timeout=60 \
          -x Log_level=Err
# scswitch -Z -g rg1
# scstat -g
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`SUNWscgds`

SEE ALSO rt_callbacks(1HA), scdsbuilder(1HA), scha_resource_get(1HA), hatimerun(1M), pmfadm(1M), scrgadm(1M), scstat(1M), scswitch(1M), signal(3HEAD), rt_reg(4), attributes(5), r_properties(5), scalable_service(5)

NAME

SUNW.HAStorage, HAStorage – resource type to synchronize action between HA storage and data services

DESCRIPTION

`SUNW.HAStorage` describes a resource type that defines resources in a resource group to synchronize the actions between the cluster file system, global devices, and relevant data services.

There is no direct synchronization between resource groups and disk device groups (and the cluster file system). As a result, during a cluster reboot or failover, an attempt to start a data service can occur while its dependent global devices or cluster file systems are still unavailable. Consequently, the data service's `START` method might timeout and the service is not started on the cluster.

`SUNW.HAStorage` is a resource type that specifically monitors the storage device services. You add a resource of this type to resource groups containing other resources and set up dependencies between the other resources and the `HAStorage` resource. The `HAStorage` resource continually tests the availability of the global devices, device groups, and the cluster file system. The dependencies ensure that the data service resources does not attempt to start until the device services are available.

When a data service resource is set up with a "strong dependency" upon a `SUNW.HAStorage` resource, the data service resources are not started before all dependent global devices and cluster file systems become available.

Multiple `SUNW.HAStorage` resources can be set up within a cluster to obtain finer granularity of the service monitoring checks. Device services that the data service needs to check and wait for but not depend upon to be online can be defined in a separate resource, and a "weak dependency" can be set up from the data resource to the device resource.

In this case, the data service resource waits for the resource to check if the device services are all available. If not, even if the `SUNW.HAStorage` `START` method times out, the data service can still be brought online. This feature is useful to some data services. For example, assume a Web server depends on ten cluster file systems. If only one file system isn't ready within the timeout period, the Web service should still go online since it still can provide 90 percent of the services.

Two extension properties are associated with the `SUNW.HAStorage` resource type: `ServicePaths` and `AffinityOn`.

`ServicePaths` Contains valid global device group names, paths to global devices, or cluster file system mount points that are to be checked. They are defined in the format of

paths`[,...]`.

A typical example of a global device group is `nfs-dg`. A path to a global device is a valid device path in the global device namespace, such as `/dev/global/dsk/d5s2`, `/dev/global/dsk/d1s2`, or `/dev/global/rmt/0`. A cluster file system mount point is a valid global mount point defined in `/etc/vfstab` on all cluster nodes

of the cluster. You can define a global device group, a global device path, and a cluster file system mount point in one `SUNW.HAStorage` resource.

`AffinityOn` A boolean flag that specifies whether the `SUNW.HAStorage` resource needs to do an affinity switchover for the global devices and cluster file systems defined in `ServicePaths`.

When `AffinityOn` is set to `False`, the `SUNW.HAStorage` resource passively waits for the specified global services to become available. As a result, the primary of each online global service might not be the same node that is the primary of the resource group.

The purpose of an affinity switchover is to enhance performance by having data services and their dependent global services run on the same node. For each global service, the `SUNW.HAStorage` resource attempts affinity switchover only once. If switchover fails, nothing is affected and the availability check occurs normally.

The default value for `ServicePaths` is the empty string. The default value for `AffinityOn` is `True`. Both extension properties can be changed at any time when the resource group is offline.

For scalable service resources, the setting of the `AffinityOn` flag is ignored and no affinity switchover can be done. There is no benefit to switching over the disk device services because the scalable data service can be running on multiple nodes simultaneously.

SEE ALSO `rt_reg(4)`

NOTES `SUNW.HAStorage` specifies resources that check and wait for the specified global devices, device group, and cluster file systems to become available. The checking is only meaningful when data service resources (application resources) in the same resource group are set up with the correct dependency upon the `SUNW.HAStorage` resources. Otherwise, no synchronization is done.

Avoid configuring two different `SUNW.HAStorage` resources in different resource groups with their ServicePaths property referencing the same global resource and with both `AffinityOn` flags set to `True`. When the cluster is booting or during a switchover, the resource groups might end up mastered on two different nodes. Both of the `SUNW.HAStorage` resources would attempt to do an affinity switchover of the same device group, resulting in a race condition. In this case, redundant switchovers would occur and the device group might not end up being mastered by the most preferred node.

The waiting time for global services to become available is specified by the `Prenet_Start_Timeout` property in `SUNW.HAStorage`. The time is tunable with a default value of 30 minutes (1,800 seconds).

NAME

SUNW.HAStoragePlus – Resource type to enforce dependencies between Sun Cluster device services/file systems and data services.

DESCRIPTION

`SUNW.HAStoragePlus` describes a resource type which allows for specifying dependencies between data service resources and device groups, cluster (global) and local file systems. This enables data services to be brought online only after their dependent device groups and file systems are guaranteed to be available. `HAStoragePlus` also provides support for mounting, unmounting and checks of file systems.

Resource groups by themselves do not provide for direct synchronization with disk device groups, cluster or local file systems. As a result, during a cluster reboot or failover, an attempt to start a data service can occur while its dependent global devices, and file systems are still unavailable. Consequently, the data service's `START` method might timeout resulting in data service failure.

`SUNW.HAStoragePlus` represents the device groups, cluster and local file systems which are to be used by one or more data service resources. One adds a resource of type `SUNW.HAStoragePlus` to a resource group and sets up dependencies between other resources and the `SUNW.HAStoragePlus` resource. These dependencies ensure that the data service resources are brought online after:

1. All specified device services are available (and collocated if necessary)
2. All specified file systems are mounted following their checks

The `FilesystemMountPoints` extension property allow for the specification of either global or local file systems, that is, file systems that are either accessible from all nodes of a cluster or from a single cluster node. Local file systems managed by a `SUNW.HAStoragePlus` resource are mounted on a single cluster node and require the underlying devices to be Sun Cluster global devices. `SUNW.HAStoragePlus` resources specifying local file systems can only belong in a failover resource group with affinity switchovers enabled. These local file systems can therefore be termed failover file systems. Both local and global file system mount points can be specified together.

A file system whose mount point is present in the `FilesystemMountPoints` extension property is assumed to be local if its `/etc/vfstab` entry satisfies both of the following conditions:

1. Non global mount option
2. Mount at boot flag is set to no

Note – Instances of the `SUNW.HAStoragePlus` resource type ignore the mount at boot flag for global file systems.

Four extension properties are associated with the `SUNW.HAStoragePlus` resource type:

GlobalDevicePaths Contains a list of valid global device group names or global device paths. They are defined in the format of paths[,...]. Default is an empty list.

FilesystemMountPoints Contains a list of valid file system mount points. They are defined in the format of paths[,...]. Default is an empty list. Each file system mount point should have an equivalent `/etc/vfstab` entry across all cluster nodes.

AffinityOn A Boolean flag that specifies whether the `SUNW.HAStoragePlus` resource needs to do an affinity switchover for all global devices defined in the `GlobalDevicePaths` and `FilesystemMountPoints` extension properties. Affinity switchover is set by default, that is, `AffinityOn` is set to `TRUE`.

When `AffinityOn` is set to `FALSE`, the `SUNW.HAStoragePlus` resource passively waits for the specified global services to become available. In this case, the primary of each online global device service might not be the same node which is the primary of the resource group.

The purpose of an affinity switchover is to enhance performance by ensuring the colocation of the device and resource groups on a specific node. Data reads and writes therefore will always occur over the device primary paths. Affinity switchovers require the potential primary list for the resource group and the node list for the device groups to be equivalent. The `SUNW.HAStoragePlus` resource performs an affinity switchover for each device service only once, that is, when the `HastoragePlus` resource is brought online.

The setting of the `AffinityOn` flag is ignored for scalable services. Affinity switchovers are not possible with scalable resource groups.

FilesystemCheckCommand `SUNW.HAStoragePlus` conducts a file system check on each unmounted file system before attempting to mount it. The default file system check command is `/usr/sbin/fsck -o p` for UFS and VxFS filesystems, and `/usr/sbin/fsck` for other file systems. The `FilesystemCheckCommand` extension property can be used to override this default file

system check specification and instead specify an alternate command string/executable. This command string/executable will then be invoked on all unmounted file systems.

The default `FilesystemCheckCommand` extension property value is `NULL`. When the `FilesystemCheckCommand` is set to `NULL` the command will be assumed to be `/usr/sbin/fsck -o p` for UFS/VxFS filesystems and `/usr/sbin/fsck` for other file systems. When the `FilesystemCheckCommand` is set to a user specified command string, `SUNW.HAStoragePlus` will elect to invoke this command string with the file system mount point as an argument. Any arbitrary executable can be specified in this manner. A non-zero return value will be treated as a error which occured during the file system check operation, causing the start method to fail. Any arbitrary executable can be specified in this manner. When the `FilesystemCheckCommand` is set to `/bin/true`, file system checks will altogether be avoided.

SEE ALSO `rt_reg(4)`, `SUNW.HAStorage(5)`

NOTES The `HAStoragePlus` RT is a part of the `SUNWscu` package.

Data service resources within a given resource group should be made dependent on a `SUNW.HAStoragePlus` resource. Otherwise, no synchronization is possible between the data services and the global devices/file systems. Strong resource dependencies ensure that the `SUNW.HAStoragePlus` resource is brought online before other resources are brought online. Local file systems managed by SUNW.HAStorage resource are mounted only when the resource is brought online.

Although unlikely, the `SUNW.HAStoragePlus` resource is capable of mounting any global file system found to be in a un mounted state. It is recommended that `UFS` file systems have logging enabled.. All file systems are mounted in the overlay mode. Local file systems will be forcibly unmounted.

Avoid configuring multiple `SUNW.HAStoragePlus` resources in different resource groups referring to the same device group(s) and with `AffinityOn` flags set to `TRUE`. Redundant device switchovers could occur resulting in the dislocation of resource and device groups.

The waiting time for all device services and file systems to become available is specified by the `Prenet_Start_Timeout` property in `SUNW.HAStoragePlus`. This is a tunable parameter.

NAME

SUNW.rac_cvm, rac_cvm – resource type implementation that represents the VERITAS Volume Manager (VxVM) component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_cvm` resource type represents the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can use the `SUNW.rac_cvm` resource type to represent this component *only* if the cluster feature of VxVM is enabled.

Instances of the `SUNW.rac_cvm` resource type hold VxVM component configuration parameters. Instances of this type also show the status of a reconfiguration of the VxVM component.

The `SUNW.rac_cvm` resource type is a single-instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup(1M)` utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm(1M)` utility

You can set the following extension properties of the VxVM component resource by using the `scrgadm` utility.

Note – Some extension properties are tunable only when the resource is disabled. You can modify such extension properties only when VxVM is *not* running in cluster mode on any cluster node.

`Cvm_abort_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 40. This property specifies the timeout (in seconds) for the abort step of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_return_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 40. This property specifies the timeout (in seconds) for the return step of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_start_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the start step of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_step1_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 1 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_step2_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 2 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_step3_timeout`
Type integer; minimum 30; maximum 99999; defaults to 240. This property specifies the timeout (in seconds) for step 3 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_step4_timeout`
Type integer; minimum 100; maximum 99999; defaults to 320. This property specifies the timeout (in seconds) for step 4 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Cvm_stop_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 40. This property specifies the timeout (in seconds) for the stop step of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time. The modified value is used for the next reconfiguration of the VxVM component.

`Reservation_timeout`
Type integer; minimum 100; maximum 99999; defaults to 325. This property specifies the timeout (in seconds) for the reservation step of a reconfiguration of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Vxclust_num_ports`
Type integer; minimum 16; maximum 64; defaults to 32. This property specifies the number of communications ports that the `vxclust` program uses. You can modify this property only when the resource is disabled. The modified value is used for the next reconfiguration of the VxVM component.

`Vxclust_port`
Type integer; minimum 1024; maximum 65535; defaults to 5568. This property specifies the communications port number that the `vxclust` program uses. You can modify this property only when the resource is disabled. The modified value is used for the next reconfiguration of the VxVM component.

`Vxconfigd_port`
Type integer; minimum 1024; maximum 65535; defaults to 5560. This property specifies the communications port number that the VxVM component configuration daemon `vxconfigd` uses. You can modify this property only when the resource is disabled. The modified value is used for the next reconfiguration of the VxVM component.

`Vxkmsgd_port`
Type integer; minimum 1024; maximum 65535; defaults to 5559. This property specifies the communications port number that the VxVM component messaging daemon `vxkmsgd` uses. You can modify this property only when the resource is disabled. The modified value is used for the next reconfiguration of the VxVM component.

EXAMPLES

EXAMPLE 1 Changing a Property of a `rac_cvm` Resource

This example sets the timeout for step 4 of a reconfiguration of the VxVM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters to 300 seconds. The example assumes that an instance of the `SUNW.rac_cvm` resource type named `rac_cvm` has been created.

```
example# scrgadm -c -j rac_cvm\\
         -x cvm_step4_timeout=300
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcvm

SEE ALSO

`scrgadm`(1M), `scsetup`(1M), `attributes`(5)

NAME

SUNW.rac_framework, rac_framework – resource type implementation for the framework that enables Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_framework` resource type represents the framework that enables Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. This resource type enables you to monitor the status of this framework.

The `SUNW.rac_framework` resource type is a single instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup`(1M) utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm`(1M) utility

The Sun Cluster Support for Oracle Parallel Server/Real Application Clusters framework resource has no extension properties.

EXAMPLES

EXAMPLE 1 Creating a `rac_framework` Resource

This example registers the `SUNW.rac_framework` resource type and creates an instance of the `SUNW.rac_framework` resource type named `rac_framework`. The example assumes that a resource group named `rac-framework-rg` has been created.

```
example# scrgadm -a -t SUNW.rac_framework
example# scrgadm -a -j rac_framework \
         -g rac-framework-rg \
         -t SUNW.rac_framework
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscucm

SEE ALSO

`scrgadm`(1M), `scsetup`(1M), `attributes`(5)

NAME

SUNW.rac_hwraid, rac_hwraid – resource type implementation that represents the hardware redundant array of independent disks (RAID) component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_hwraid` resource type represents the hardware RAID component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters.

The `SUNW.rac_hwraid` resource type is a single-instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup`(1M) utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm`(1M) utility

You can set the following extension properties of the hardware RAID resource by using the `scrgadm` utility.

`Reservation_timeout`
Type integer; minimum 100; maximum 99999; defaults to 325. This property specifies the timeout (in seconds) for the reservation step of a reconfiguration of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

EXAMPLES

EXAMPLE 1 Changing a Property of a `rac_hwraid` Resource

This example sets the timeout for the reservation step of a reconfiguration of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters to 350 seconds. The example assumes that an instance of the `SUNW.rac_hwraid` resource type named `rac_hwraid` has been created.

```
example# scrgadm -c -j rac_hwraid\
         -x reservation_timeout=350
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWhwraid

SEE ALSO

`scrgadm`(1M), `scsetup`(1M), `attributes`(5)

NAME

SUNW.rac_svm, rac_svm – resource type implementation that represents the Solaris Volume Manager component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_svm` resource type represents the Solaris Volume Manager for Sun Cluster component of the Sun Cluster framework for Oracle Parallel Server/Real Application Clusters.

Instances of the `SUNW.rac_svm` resource type hold Solaris Volume Manager for Sun Cluster component configuration parameters. Instances of this type also show the status of a reconfiguration of the Solaris Volume Manager for Sun Cluster component.

The `SUNW.rac_svm` resource type is a single-instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup(1M)` utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm(1M)` utility

You can set the following extension properties of the Solaris Volume Manager for Sun Cluster component resource by using the `scrgadm` utility.

`Debug_level`
Type integer; minimum 0; maximum 10; defaults to 1. This property specifies the debug level for the Solaris Volume Manager for Sun Cluster module of Sun Cluster framework for Oracle Parallel Server/Real Application Clusters. When the debug level is increased, more messages are written to the log files during reconfiguration. You can modify this property at any time.

`Reservation_timeout`
Type integer; minimum 100; maximum 99999; defaults to 325. This property specifies the timeout (in seconds) for the reservation step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_abort_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the abort step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_return_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the return step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_start_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the start step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_step1_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for step 1 of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_step2_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for step 2 of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_step3_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for step 3 of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_step4_timeout`
Type integer; minimum 100; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for step 4 of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

`Svm_stop_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 120. This property specifies the timeout (in seconds) for the stop step of a reconfiguration of the Solaris Volume Manager for Sun Cluster module of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. You can modify this property at any time.

EXAMPLES

EXAMPLE 1 Changing a Property of a `rac_svm` Resource

This example sets the timeout for step 4 of a reconfiguration of the Solaris Volume Manager for Sun Cluster component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters to 300 seconds. The example assumes that an instance of the `SUNW.rac_svm` resource type named `rac_svm` has been created.

```
example# scrgadm -c -j rac_svm \
         -x svm_step4_timeout=300
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWscmd

SEE ALSO `attributes`(5)

`scrgadm`(1M), `scsetup`(1M)

NAME

SUNW.rac_udlm, rac_udlm – resource type implementation for the configuration of the UNIX Distributed Lock Manager (Oracle UDLM) component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters

DESCRIPTION

The `SUNW.rac_udlm` resource type enables the management of the Oracle UDLM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters. The management of this component involves the following activities:

- Setting the parameters of the Oracle UDLM component
- Monitoring the status of the Oracle UDLM component

The `SUNW.rac_udlm` resource type is a single-instance resource type. Only one resource of this type may be created in the cluster.

To register this resource type and create instances of this resource type, use one of the following utilities:

- The `scsetup(1M)` utility, specifying the option for configuring Sun Cluster Support for Oracle Parallel Server/Real Application Clusters
- The `scrgadm(1M)` utility

You can set the following extension properties for an Oracle UDLM resource by using the `scrgadm` utility.

Note – Some extension properties are tunable only when the resource is disabled. You can modify such extension properties only when the Oracle UDLM is *not* running on any cluster node.

`Failfastmode`
Type enum; defaults to `panic`. This property specifies the failfast mode of the node on which the Oracle UDLM is running. The failfast mode determines the action that is performed in response to a critical problem with this node. The possible values of this property are as follows:

`off`	Failfast mode is disabled.
`panic`	The node is forced to panic.

You can modify this property at any time. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Num_ports`
Type integer; minimum 16; maximum 64; defaults to 32. This property specifies the number of communications ports that the Oracle UDLM uses. You can modify this property only when the resource is disabled. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Oracle_config_file`
Type string; defaults to `/etc/opt/SUNWcluster/conf/udlm.conf`. This property specifies the configuration file that the Oracle distributed lock manager

(DLM) uses. This file must already exist. The file is installed when the Oracle software is installed. For more information, refer to the documentation for the Oracle software. You can modify this property at any time. The modified value is used for the next start-up of the Oracle DLM.

`Port`
Type integer; minimum 1024; maximum 65500; defaults to 6000. This property specifies the communications port number that the Oracle UDLM uses. You can modify this property only when the resource is disabled. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Schedclass`
Type enum; defaults to `RT`. This property specifies the scheduling class of the Oracle UDLM that is passed to the `priocntl`(1) command. The possible values of this property are as follows:

`RT` Real-time

`TS` Time-sharing

`IA` Interactive

You can modify this property only when the resource is disabled. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Schedpriority`
Type integer; minimum 0; maximum 59; defaults to 11. This property specifies the scheduling priority of the Oracle UDLM that is passed to the `priocntl` command. You can modify this property only when the resource is disabled. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Udlm_abort_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 325. This property specifies the timeout (in seconds) for the abort step of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_start_step_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for the start step of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next start-up of the Oracle UDLM. The Oracle UDLM is started when a node is rebooted.

`Udlm_step1_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 1 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_step2_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 2 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_step3_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 3 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_step4_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 4 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

`Udlm_step5_timeout`
Type integer; minimum 30; maximum 99999; defaults to 100. This property specifies the timeout (in seconds) for step 5 of an Oracle UDLM reconfiguration. You can modify this property at any time. The modified value is used for the next reconfiguration of the Oracle UDLM.

EXAMPLES

EXAMPLE 1 Changing a Property of a `rac_udlm` Resource

This example sets the timeout for step 4 of a reconfiguration of the Oracle UDLM component of Sun Cluster Support for Oracle Parallel Server/Real Application Clusters to 45 seconds. The example assumes that an instance of the `SUNW.rac_udlm` resource type named `rac_udlm` has been created.

```
example# scrgadm -c -j rac_udlm\
         -x udlm_step4_timeout=45
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWudlm

SEE ALSO

`priocntl`(1), `scrgadm`(1M), `scsetup`(1M), `attributes`(5)

NAME

SUNW.RGOffload, RGOffload – resource type to offload specified resource groups

DESCRIPTION

`SUNW.RGOffload` describes a resource type that allows resources configured in failover resource groups to offload other specified resource groups.

This facility is most useful when the limited resources on cluster nodes prevent multiple data services from running simultaneously on a node. In such situations, a `RGOffload` resource in a resource group containing critical data services is configured to offload other resource groups.

You can use the `scrgadm(1M)` command or resource configuration GUI to add a `RGOffload` resource to the resource group containing critical data service resources, setup dependencies of the critical data service resources on this resource, and configure the resource groups to be offloaded from a node when critical data service resources are running on it. The dependencies ensure that the data service resources do not attempt to start on a node until the `START` method of the `RGOffload` resource has offloaded, or at least attempted to offload the specified resource groups from the node.

Resource groups specified to be offloaded must have their Desired_primaries property set to 0. The fault monitor of the `SUNW.RGOffload` resource will attempt to keep such resource groups online on as many healthy nodes as possible, limited by the Maximum_primaries property of individual resource groups. The fault monitor checks the status of specified resource groups on all nodes every `Thorough_probe_interval`.

When a data service resource is set up with a "strong dependency" upon a `SUNW.RGOffload` resource, the data service resource is not started on a node if there is a failure in offloading specified resource groups from that node. A data service resource set up with a "weak dependency" upon the `SUNW.RGOffload` resource may start when specified resource groups cannot be successfully offloaded from the node. An attempt would be made to offload the specified resource groups, but a failure in doing so will not prevent the startup of the data service resource.

See `r_properties(5)` for a complete description of the standard resource properties.

Extension Properties

`Monitor_retry_count`
Type integer; defaults to `4`. This property controls fault-monitor restarts. The property indicates the number of times that the process monitor facility (PMF) restarts the fault monitor. The property corresponds to the `-n` option passed to the `pmfadm(1M)` command. The RGM counts the number of restarts in a specified time window (see the property `Monitor_retry_interval`). Note that this property refers to the restarts of the fault monitor itself, not the `SUNW.RGOffload` resource. You can modify the value for this property at any time.

`Monitor_retry_interval`
Type integer; defaults to `2`. This property indicates the time window in minutes during which the RGM counts fault-monitor failures. The property corresponds to the `-t` option passed to the `pmfadm(1M)` command. If the number of times that the fault monitor fails exceeds the value of the extension property

Monitor_retry_count, the PMF does not restart the fault monitor. You can modify the value for this property at any time.

rg_to_offload
Type string array, specified as a comma-separated list of resource groups. No default exists for this field. You must provide the value when creating the resource. This property indicates the list of resource groups to be offloaded. All resource groups in this property must have Desired_primaries set to 0. rg_to_offload should not contain the resource group in which the RGOffload resource is being configured. rg_to_offload should also not contain resource groups dependent upon each other. For example, if resource group RG-B depends on resource group RG-A, then both, RG-A and RG-B should not be configured in this extension property. SUNW.RGOffload resource type does not check for dependencies among resource groups in the rg_to_offload extension property. You can modify the value of this property at any time.

continue_to_offload
Type boolean; defaults to TRUE. This property indicates whether to continue offloading the next resource group in the list specified in the rg_to_offload property in case of error in offloading any resource group. You can modify the value of this property at any time.

max_offload_retry
Type integer; defaults to 15. This property indicates the number of attempts during the startup of RGOffload resource to offload a resource group specified in the rg_to_offload property if there is a failure due to cluster or resource group reconfiguration. This value applies to all resource groups in the rg_to_offload property. When the value of this property is greater than 0, successive attempts to offload the same resource group would be made after approximately 10 second intervals. You can modify the value of this property at any time.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWrgofl

SEE ALSO

pmfadm(1M), scha_resource_get(1HA), scrgadm(1M), scswitch(1M), scha_cluster_get(3HA), scha_resourcegroup_get(3HA) , attributes(5), r_properties(5)

Sun Cluster Data Services Installation and Configuration Guide

SC31 7

NAME

clprivnet – SUNW,clprivnet Sun Cluster private network driver

SYNOPSIS

```
/dev/clprivnet
```

DESCRIPTION

The `SUNW,clprivnet` Sun Cluster private network driver is a STREAMS pseudo driver supporting Sun Cluster resident applications that use standard Solaris interfaces to communicate over the Sun Cluster private network. By striping data traffic over all links, this driver optimally utilizes the bandwidth of the private network while supporting highly available, software fault-tolerant communication.

APPLICATION PROGRAMMING INTERFACE

The driver is supported by the character-special device `/dev/clprivnet`, but is reserved for Sun Cluster internal operation and the standard Solaris network utilities. This interface must not be directly used for general application communication.

ADMINISTRATION

The administration and configuration of the driver as a network interface is done completely by the Sun Cluster infrastructure internals.

FILES

`/dev/clprivnet`
`clprivnet` special character device

`/usr/kernel/drv/clprivnet.conf`
System-wide default device driver properties

NAME

did – user configurable disk id driver

DESCRIPTION

Disk ID (DID) is a user configurable pseudo device driver that provides access to underlying disk, tape, and CDROM devices. When the device supports unique device ids, multiple paths to a device are determined according to the device id of the device. Even if multiple paths are available with the same device id, only one DID name is given to the actual device.

In a clustered environment, a particular physical device will have the same DID name regardless of its connectivity to more than one host or controller. This, however, is only true of devices that support a global unique device identifier such as physical disks.

DID maintains parallel directories for each type of device that it manages under `/dev/did`. The devices in these directories behave the same as their non-DID counterparts. This includes maintaining slices for disk and CDROM devices as well as names for different tape device behaviors. Both raw and block device access is also supported for disks by means of `/dev/did/rdsk` and `/dev/did/rdsk`.

At any point in time, I/O is only supported down one path to the device. No multipathing support is currently available through DID.

Before a DID device can be used, it must first be initialized by means of the `scdidadm(1M)` command.

IOCTLS

The DID driver maintains an admin node as well as nodes for each DID device minor.

No user ioctls are supported by the admin node.

The `DKIOCINFO` ioctl is supported when called against the DID device nodes such as `/dev/did/rdsk/d0s2`.

All other ioctls are passed directly to the driver below.

FILES

`/dev/did/dsk/dnsm`	block disk or CDROM device, where `n` is the device number and `m` is the slice number
`/dev/did/rdsk/dnsm`	raw disk or CDROM device, where `n` is the device number and `m` is the slice number
`/dev/did/rmt/n`	tape device , where `n` is the device number
`/dev/did/admin`	administrative device
`/kernel/drv/did`	driver module
`/kernel/drv/did.conf`	driver configuration file
`/etc/did.conf`	`scdidadm` configuration file for non-clustered systems
`Cluster Configuration Repository` (CCR) files	`scdidadm(1M)` maintains configuration in the CCR for clustered systems

SEE ALSO `devfsadm`(1M), `scdidadm`(1M)

NOTES DID creates names for devices in groups, in order to decrease the overhead during device hot-plug. For disks, device names are created in `/dev/did/dsk` and `/dev/did/rdsk` in groups of 100 disks at a time. For tapes, device names are created in `/dev/did/rmt` in groups of 10 tapes at a time. If more devices are added to the cluster than are handled by the current names, another group will be created.

SC31 7p

NAME sctransp_dlpi – configure the dlpi cluster interconnect

DESCRIPTION `dlpi` is a supported cluster transport type.

SEE ALSO `scconf`(1M), `scinstall`(1M)

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W