Sun Cluster 3.1 Reference Manual

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

Part No: 816–5251–10
February 2003, Revision A

Adobe PostScript

021002@4660

Contents

Preface

Both novice users and those familar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question "What does it do?" The man pages in general comprise a reference manual. They are not intended to be a tutorial.

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 needed to write device drivers in the kernel environment. It 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 used by drivers to share information between the driver and the kernel.

Below 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 there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man`(1) for more information about man pages in general.

NAME	This section gives the names of the commands or functions documented, followed by a brief description of what they do.
SYNOPSIS	This section shows the syntax of commands or functions. When 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 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, such that everything enclosed must be treated as a unit.

PROTOCOL	This section occurs only in subsection 3R to indicate the protocol description file.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It 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 of which have an `io` ending, such as `mtio`(7I).
OPTIONS	This secton 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 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` indicating 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 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 including 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 for various error conditions.
FILES	This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each 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 causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. 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*`]`

`scdsconfig` `-s` *start_command* `[-u` *start_method_timeout*`]` `[-t` *stop_command*`]` `[-v` *stop_method_timeout*`]` `[-d` *working_directory*`]`

DESCRIPTION

The `scdsconfig` command configures the resource type template previously created by `scdscreate`(1HA). There are two forms of this command. Use the first form for network aware (client-server model) applications. Use the second form for non-network aware (clientless) applications. See `SYNOPSIS`.

`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; alternatively, you can specify that directory using the `-d` option. `scdsconfig` configures the resource type template by plugging the user specified parameters at appropriate places in the generated code. If C was the type of generated source code, it compiles the code. `scdsconfig` puts everything into an installable Solaris package. It 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 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 OK. An exit status of greater than `0` indicates a failure of the application to perform correctly. In this event, the resources of this resource type will either be restarted on the same node or the resource group containing the resource will be 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. It 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 necessary to start the application. To create a resource type with multiple independent process trees, you specify a text file containing 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 then the generated code stops the application via signals. The stop command is allotted 80% of the timeout value to stop the application. If it fails to stop it within this period a `SIGKILL` is allotted 15% of the timeout value to stop the application. If that 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`	Successful completion.
`>0`	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

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

NAME

scdscreate – create a Sun Cluster resource type template

SYNOPSIS

scdscreate -V *vendor_id* -T *resource_type_name* [-s] [-d *working_directory*] [-k | -g]

scdscreate -V *vendor_id* -T *resource_type_name* -a [-s] [-d *working_directory*] [-k | -g]

DESCRIPTION

The `scdscreate` command creates a template for making an application highly available (HA) or scalable. Use the first form for network aware (client-server model) applications. Use the second form for non-network aware (clientless) applications. See `SYNOPSIS`.

The template can be created in two fundamentally different models:

Generic Data Service (`GDS`)	`scdscreate` creates a set of three *driving scripts* which work off of a single resource type `SUNW.gds` pre-installed on the cluster. These scripts will be named as *start{RT_Name}*, *stop{RT_Name}*, and *remove{RT_Name}* and will start, stop and remove an instance of that application respectively. In this model the implementation of the `SUNW.gds` resource type 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*, to store the user supplied configuration information for the resource type. `scdscreate` allows the creation of one resource type per directory; different resource types must be created in different directories.

OPTIONS

The following options are supported:

`-a`	This argument specifies that the resource type being created is not network aware. `scdscreate` disables all the networking related code in the template created. Note that it is not possible to enable fault probing for a non-network aware application. However, the application is started under the Process Monitor Facility (PMF), which monitors the application and restarts it if it fails to remain alive. See `pmfadm`(1M).

`-d` *working_directory*	Specify this optional argument to create 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`	Specify this optional argument to generate the *Generic Data Service* form of template to make an application HA or scalable.
`-k`	Specify this optional argument to generate source code in Korn Shell rather than C. See `ksh`(1).
`-s`	Specifies this optional argument to indicate that the resource type is scalable. It is possible to 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, in conjunction with the vendor id, uniquely identifies the resource type being created.
`-V` *vendor_id*	The vendor id is typically the stock symbol, or some other identifier of the vendor creating the resource type. `scdscreate` prefixes the vendor id, followed by a dot (`.`) to the resource type name. This ensures the uniqueness of the resource type name in the event that multiple vendors use the same name.

EXIT STATUS

`0`	Successful completion.
`>0`	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), `scdsconfig`(1HA), `pmfadm`(1M), `attributes`(5)

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. Case is not considered in recognition of the *optag* argument.

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 facility being used for the cluster log. The value may be used as the facility level of 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

SYNOPSIS

scha_command `-O` *optag*...

DESCRIPTION

The Sun Cluster `scha_cluster_get`(1HA), `scha_control`(1HA), `scha_resource_get`(1HA), `scha_resourcegroup_get`(1HA), and `scha_resourcetype_get`(1HA), commands are in shell script 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 shell 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 outputs 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. The recognition of *optag* values is case-insensitive for all of the `scha` commands.

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

Result Formats

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

Result Type	standard output Format
string	"%s\n"
int	"%d\n"
unsigned_int	"%u\n"
boolean	"TRUE\n" or "FALSE\n"
enum	"%s\n". The string name of an enum value
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 INSTALLED_NODES and GLOBAL_RESOURCES_USED properties.
unsigned_int_array	Each element in the array is output in the format "%u\n"

Result Type	**standard output Format**
extension	"%s\n". The type attribute of the extension property - one of STRING,INT, BOOLEAN, ENUM, or STRINGARRAY. Following the type inforation, 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
status	"%s\n%s\n". The first string is the status - one of the status enum values: OK, OFFLINE, FAULTED, DEGRADED, or UNKNOWN. The second string is the status message

Optag Result Types

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

optags **for** `scha_cluster_get`**(1HA)**	**Result Type**
NODENAME_LOCAL	string
NODENAME_NODEID	string
ALL_NODENAMES	string_array
NODEID_LOCAL	unsigned_int
NODEID_NODENAME	unsigned_int
ALL_NODEIDS	unsigned_int_array
PRIVATELINK_HOSTNAME_LOCAL	string
PRIVATELINK_HOSTNAME_NODE	string
ALL_PRIVATELINK_HOSTNAMES	string_array
NODESTATE_LOCAL	enum - UP, DOWN
NODESTATE_NODE	enum - UP, DOWN
SYSLOG_FACILITY	int
ALL_RESOURCEGROUPS	string_array
ALL_RESOURCETYPES	string_array
CLUSTERNAME	string

optags **for** `scha_control`**(1HA)**
GIVEOVER
RESTART
CHECK_GIVEOVER
CHECK_RESTART

optags **for** `scha_resource_get`**(1HA)**	**Result Type**
R_DESCRIPTION	string
TYPE	string
ON_OFF_SWITCH	enum - DISABLED, ENABLED
MONITORED_SWITCH	enum - DISABLED, ENABLED
RESOURCE_STATE	enum - ONLINE, OFFLINE, START_FAILED, STOP_FAILED. MONITOR_FAILED, ONLINE_NOT_MONITORED
CHEAP_PROBE_INTERVAL	int
THOROUGH_PROBE_INTERVAL	int
RETRY_COUNT	int
RETRY_INTERVAL	int
FAILOVER_MODE	enum - NONE, HARD, SOFT
RESOURCE_DEPENDENCIES	string_array
RESOURCE_DEPENDENCIES_WEAK	string_array
LOGICAL_HOSTNAMES_USED	string_array
STATUS	status
START_TIMEOUT	int
STOP_TIMEOUT	int
VALIDATE_TIMEOUT	int
UPDATE_TIMEOUT	int
INIT_TIMEOUT	int
FINI_TIMEOUT	int
BOOT_TIMEOUT	int
MONITOR_START_TIMEOUT	int

optags **for** `scha_resource_get`**(1HA)**	**Result Type**
MONITOR_STOP_TIMEOUT	int
MONITOR_CHECK_TIMEOUT	int
PRENET_START_TIMEOUT	int
POSTNET_STOP_TIMEOUT	int
STATUS_NODE	status
RESOURCE_STATE_NODE	enum - see RESOURCE_STATE for values
EXTENSION	extension
ALL_EXTENSIONS	string_array
GROUP	string

optags **for** `scha_resource_get`**(1HA) and** `scha_resourcetype_get`**(1HA)**	**Result Type**
RT_DESCRIPTION	string
RT_BASEDIR	string
SINGLE_INSTANCE	boolean
INIT_NODES	enum - RG_PRIMARIES, RT_INSTALLED_NODES
INSTALLED_NODES	string_array. An asterisk is returned to indicate all nodes.
FAILOVER	boolean
API_VERSION	int
RT_VERSION	string
PKGLIST	string_array
START	string
STOP	string
VALIDATE	string
UPDATE	string
INIT	string
FINI	string
BOOT	string
MONITOR_START	string

optags **for** `scha_resource_get`**(1HA) and** `scha_resourcetype_get`**(1HA)**	**Result Type**
MONITOR_STOP	string
MONITOR_CHECK	string
PRENET_START	string
POSTNET_STOP	string
IS_LOGICAL_HOSTNAME	boolean
IS_SHARED_ADDRESS	boolean

optags **for** `scha_resourcegroup_get`**(1HA)**	**Result Type**
RG_DESCRIPTION	string
NODELIST	string_array
MAXIMUM_PRIMARIES	int
DESIRED_PRIMARIES	int
FAILBACK	boolean
RESOURCE_LIST	string_array
RG_STATE	enum - UNMANAGED, ONLINE, OFFLINE, PENDING_ONLINE, PENDING_OFFLINE, ERROR_STOP_FAILED
RG_DEPENDENCIES	string_array
GLOBAL_RESOURCES_USED	string_array. An asterisk is returned to indicate all resources.
LOGICAL_HOST	boolean
PATHPREFIX	string
RG_STATE_NODE	enum - see RG_STATE for values

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_calls(3HA), 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. The 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.

OPTIONS

The following options are supported:

`-G` *group*
: 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*
: Request *optag* options.

The following *optag* values are supported:

`RESTART`
request that the resource group named by the `-G` option 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.

`RESOURCE_RESTART`
request 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 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 and/or POSTNET_STOP methods. Only the MONITOR_STOP, STOP, START, and MONITOR_START methods will be applied to the resource. Network address resources on which this resource implicitly depends will not be restarted and will remain online.

RESOURCE_IS_RESTARTED

request 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 will be reflected in subsequent NUM_RESOURCE_RESTARTS queries of scha_resource_get.

If the resource's type fails to declares 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).

GIVEOVER

request 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 may be taken offline on the local node without being brought online elsewhere. The request may 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.

In addition, the resources in the group might contain `MONITOR_CHECK` callback methods that the RGM invokes in the event of a giveover request. These methods verify that a node is healthy enough to run the resource. For example, a `MONITOR_CHECK` method might verify that an essential configuration file is available on the node, and if it isn't, veto the giveover request.

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

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

The `CHECK_GIVEOVER` and `CHECK_RESTART` *optags* 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, then 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* — 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.

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

`pmfadm`(1M), `rt_callbacks`(1HA), `scha_resource_get`(1HA), `scha_cmds`(1HA), `scha_control`(3HA), `scha_calls`(3HA), `attributes`(5), `r_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. The resource information that can be accessed includes properties of the resource's type.

`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 may be stored in shell variables and parsed using shell facilities or `awk`(1) for further use by the script.

OPTIONS

The following options are supported:

`-G` *group*
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, so it is recommended.

`-O` *optag*
The *optag* argument indicates the information to be accessed. Depending on the *optag*, an additional operand may be needed to indicate the cluster node for which information is to be retrieved. Case is not considered in recognition of the *optag* arguments.

The following *Optag* values supported:

The following *optags* 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)).

```
R_DESCRIPTION
TYPE
ON_OFF_SWITCH
MONITORED_SWITCH
RESOURCE_STATE
CHEAP_PROBE_INTERVAL
THOROUGH_PROBE_INTERVAL
RETRY_COUNT
RETRY_INTERVAL
FAILOVER_MODE
RESOURCE_DEPENDENCIES
RESOURCE_DEPENDENCIES_WEAK
NETWORK_RESOURCES_USED
SCALABLE
PORT_LIST
LOAD_BALANCING_POLICY
```

```
LOAD_BALANCING_WEIGHTS
AFFINITY_TIMEOUT
WEAK_AFFINITY
UDP_AFFINITY
STATUS
START_TIMEOUT
STOP_TIMEOUT
VALIDATE_TIMEOUT
UPDATE_TIMEOUT
INIT_TIMEOUT
FINI_TIMEOUT
BOOT_TIMEOUT
MONITOR_START_TIMEOUT
MONITOR_STOP_TIMEOUT
MONITOR_CHECK_TIMEOUT
PRENET_START_TIMEOUT
POSTNET_STOP_TIMEOUT
NUM_RG_RESTARTS
NUM_RESOURCE_RESTARTS
TYPE_VERSION
RESOURCE_PROJECT_NAME
```

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 the property followed by its value, on successive lines. Shell scripts might need to discard the type to obtain the value, as shown in EXAMPLE below.

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 *optags* retrieve the corresponding resource type properties. The value of the named property of the resource's type is output. For descriptions of resource type properties, see `rt_properties`(5).

```
RT_DESCRIPTION
RT_BASEDIR
SINGLE_INSTANCE
INIT_NODES
INSTALLED_NODES
FAILOVER
API_VERSION
RT_VERSION
PKGLIST
START
```

```
STOP
VALIDATE
UPDATE
INIT
FINI
BOOT
MONITOR_START
MONITOR_STOP
MONITOR_CHECK
PRENET_START
POSTNET_STOP
IS_LOGICAL_HOSTNAME
IS_SHARED_ADDRESS
RESOURCE_PROJECT_NAME
```

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

EXAMPLES

EXAMPLE 1 A Sample Script Using `scha_resource_get`

The following script is passed `-R` and `-G` arguments giving the needed resource name and resource group name, as well as resource type. 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

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

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	Evolving

SEE ALSO

`awk`(1), `sh`(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.

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_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 in the format of 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.

OPTIONS

The following options are supported:

`-G` *group* The name of the resource group.

`-O` *optag* The *optag* argument indicates the information to be accessed. Depending on the the *optag*, an additional operand may be needed to indicate the cluster node for which information is to be retrieved. Case is not considered in recognition of the *optag* arguments.

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.

```
RG_DESCRIPTION
NODELIST
MAXIMUM_PRIMARIES
DESIRED_PRIMARIES
FAILBACK
RESOURCE_LIST
RG_STATE
RG_DEPENDENCIES
GLOBAL_RESOURCES_USED
RG_MODE
IMPLICIT_NETWORK_DEPENDENCIES
PINGPONG_INTERVAL
PATHPREFIX
RG_PROJECT_NAME
RG_STATE_NODE
```

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 giving the needed 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`	Successful completion.
`non-0`	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), `sh`(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` 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. 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.

OPTIONS

The following options are supported:

`-G` *group*	The resource group containing the resource.
`-m` *msg*	The *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*	The value of *status* can be `OK`, `DEGRADED`, `FAULTED`, `UNKNOWN`, or `OFFLINE`.

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).

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 ALLSO

`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 may be stored in shell variables and parsed using shell facilities or `awk`(1) for further use by the script.

OPTIONS

The following options are supported:

`-O` *optag* The *optag* argument indicates the information to be accessed. Case is not considered in recognition of the *optag* argument.

Optag values are listed below. The following *optags* retrieve the corresponding resource type properties. The value of the named property of the resource's type is output.

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

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

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

`non-0` 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), `sh`(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

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

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

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. 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 forever for the processes associated with the tag to exit.

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`	Successful completion.

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

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

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. This daemon initially starts when the system comes up.

`rpc.pmfd` must be started as root 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

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. This daemon initially starts when the system comes up.

`rpc.pmfd` must be started as root 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

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 and validate Sun Cluster configurations

SYNOPSIS

`sccheck` `-bvW [-h` *nodelist*`]`

DESCRIPTION

The `sccheck`utility checks and validates the Sun Cluster configuration. Depending on the state of the node that issues the command, `sccheck` performs one of the following checks:

Pre-installation checks	When issued from a node that is not running as an active cluster member, `sccheck` runs pre-installation checks on that node. These checks ensure that the node meets the minimum requirements to be successfully installed with Sun Cluster software.
Cluster configuration checks	When issued from an active member of a running cluster, `sccheck` runs cluster-configuration checks on all cluster 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 regardless of which cluster node issues the command.

The `sccheck` utility runs the following checks.

- Verifies sufficient configured swap space
- Verifies sufficient physical memory
- Validates the `/etc/vfstab` file. Configuration check only.

If `sccheck` finds problems with the configuration, it exits with a non-zero exit status and prints one or more diagnostic error messages. You must correct all errors before you attempt to run the cluster configuration in a production environment.

OPTIONS

The following options are supported:

`-b`	Disable lengthy checks.
`-h` *node1,node2...*	Specify on which nodes `sccheck` runs the check. This option is only legal when `sccheck` is issued from a node that is an active cluster member.
`-v`	Run in verbose mode.
`-W`	Disable any warnings.

EXIT STATUS

The following exit values are returned:

`0`	Successful completion.
`non-zero`	An error occurred.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

`scinstall`(1M)

Sun Cluster 3.0 Software Guide, *Sun Cluster 3.0 Installation Guide*, *Sun Cluster 3.0 System Administration Guide*

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*`]`

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

scconf `-p [-Hv [v]]`

scconf `[-H]`

DESCRIPTION

The `scconf` utility manages the Sun Cluster software configuration. You can use `scconf` to add items to the configuration, to change properties of already configured items, and to remove items from the configuration. `scconf` can also print a listing of the current configuration.

`scconf` 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 should always be the same, regardless of the node used.

Any form of the command which would result in an update to the configuration must be run as root. Non-root users may only use `scconf` to list the configuration or print help information.

All forms of the `scconf` command accept the `-H` option. Specifying the `-H` displays help information and all other options are ignored. Help information is also printed when `scconf` is invoked without options.

For each of the three forms of the command designed to update the configuration, options are processed in the order in which they are given on the command line. All updates associated with each option must complete successfully before the next option is considered.

Basic Options

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

`-H`
: If this option is given anywhere on the command line, help information is printed and all other options are ignored. Help information is also printed if `scconf` is invoked with no options.

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

`-a`
: Specify the `add` form of the `scconf` command. It can be used to add or initialize most of the items used to define the software configuration of a Sun Cluster.

Additional options are used with `-a` to specify items to be added and their associated propterties. Any number of these additional options may be combined on the same command line, as long as they are all intended for use with the `-a` option.

`-c`
Specify the `change` form of the `scconf`command. It 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 the new properties or changes. Any number of these additional options may be combined on the same command line, as long as they are all intended for use with the `-c` option.

`-r`
Specify the `remove` form of the `scconf` command. It is used to remove items from the Sun Cluster software configuration.

Additonal options are used with `-r` to specify which items to delete from the configuration. Any number of these additional options may be combined on the same command line, as long as they are all intended for use with the `-r` option.

`-p`
Specify the `print` form of the `scconf` command. This form of the command prints a listing of the current Sun Cluster configuration, items configurable through `scconf` and their properties. This option may be combined with one or more `-v` options to print more verbose listings.

Additional Options

The following options are additional options which can be combined with one or more of the basic options described above. Refer to the `SYNOPSIS` section for additional detail on which of these options are legal with which forms of `scconf`.

The additional options are as follows:

`-A` *adapter_options*
Add, remove, or change the properties of a cluster transport adapter. The node on which the given adapter is hosted need not be active in the cluster in order for these operations to succeed. The `-A` *adapter_options* for each of the three forms of the command which accept `-A` are as follows:

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

`-A trtype=type,name=`*name*`,node=`*node*`[,`*other_options*`]`

Use the following to specify `-A` *adapter_options* for the `change` form of the command:

`-A name=`*name*`,node=`*node*`[,state=`*state*`] [,`*other_options*`]`

Use the following 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`
Specify transport type. This suboption must always be included when `-A` is used with the `add` form of the command.

Examples of transport `types` are `rsm` and `dlpi`. See `sctransp_rsm`(7P) and `sctransp_dlpi`(7P).

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

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

`state=`*state*
Change the state or 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, it's state is always set to `disabled`. By default, adding a cable (see `-m`) to any of the ports on an adapter changes the state of both the port and the adapter to `enabled`. See `-m`.

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 it's ports. To enable an adapter port, it is necessary to enable the cable to which the port is connected.

[*other_options*]
If other options are available for a particular adapter type, they may be used with `-A` in the add and change forms of the command. Refer to the individual 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 on special options.

`-B` *junction_options*
Add, remove, or change 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.

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

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

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

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

Use the following to specify `-B` *junction_options* for the `remove` form of the command:

`-B name=`*name*

The `-B` option supports the following suboptions:

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

  The suboption is used to s Ethernet hubs and SCI switches are examples of cluster transport junctions which are both of `type` switch. See `scconf_transp_jct_dolphinswitch`(1M) and `scconf_transp_jct_etherswitch`(1M).

`name=`*name*
: Specify the name of a cluster transport junction. A `name` suboption must always 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*
: Change 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`.

  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 it's ports. To enable a junction port, it is necessary to 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 the `-B` option and the `add` and `change` forms of the command.

  Refer to the individual cluster transport junction man pages (for example., `scconf_transp_jct_dolphinswitch`(1M), and `scconf_transp_jct_etherswitch`(1M)) for information on special options.

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

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

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

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

`-D` *devicegroup_options*
Add disk device groups to the configuration, to change or reset properties of already existing device groups, and to remove groups from the Sun Cluster device groups configuration. Other disk device group options (*otheroptions*) play a crucial role in adding or changing device groups and their options. Pay special attention to the type dependent disk device groups options man pages (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 new device groups or to add nodes to already existing device groups. For some device group types, it may 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` *devicegroups_options* for each of the three forms of the `scconf` command which accept `-D` are as follows:

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

`type=type`
The `type` option must be used with the `add` form of the command to indicate which type of disk device group to create (for example, `vxvm`, `sds`, `svm`, or `rawdisk`).

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

[`nodelist=`*node*[:*node*]...]
For some disk device group types, a list of potential primary nodes for the device group is required when adding a group to the cluster. Refer to the type dependent disk device group man pages 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 primary for a disk device group. However, if `preferenced` is set to `false`, the first node to access a device in the group automatically becomes primary for that group. It is not legal to use the `preferenced` option when adding nodes to an already existing device group. However, it may be given when creating the group for the first time, or with the `change` form of the command.

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

When used with the `remove` form of the command, the `nodelist` option is used to remove the indicated nodes from the device group. Only by not giving a `nodelist` at all 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`}]
As long as `preferenced` is not set to `false`, nodelists for newly created device groups describe a preferred order in which nodes will attempt to take over as primary for a disk device group.

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

It is not legal to specify `preferenced` with an `add` used to add nodes to an already established device group. In this case, the already established node preference list setting is used.

[`numsecondaries=`*integer*]
This option allows system administrators to dynamically change the desired number of secondary nodes for a device group. A device group is a HA service which requires one node to act as a primary and one or more nodes to act as secondaries. The secondary nodes of a device group are able to take over and act as primary if the current primary node fails.

This integer value should be set 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` option to change the number of secondaries for a device group while maintaining a given level of availability. If a node in a device group is removed from the secondaries list, it will not be able to take over and act as a primary until it is converted back to a secondary. Before making a change to the number of secondaries, you need to assess the impact on the secondary global filesystem.

The `numsecondaries` option only applies to nodes in a device group that are currently in cluster mode and can be used together with the node preference option. If a device's `preferenced` flag is enabled, the nodes that are least preferred will be removed from the secondaries list first. If no node in a device group are flagged as preferred, the cluster will pick the node to be removed on a random basis.

When a device group's actual number of secondaries drops to less that the desired level due to node failures, nodes that were removed from the secondaries list will be 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 secondary. The conversion starts with the node in the device group with the highest preference until the number of desired secondaries is matched.

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

To set the desired number of secondaries to the system default (without having to know the default value), issue:

```
scconf -aD type=vxvm,name=foo,numsecondaries=
```

or

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

The `numsecondaries` option can only be used with `-a` option during device group creation time. The `numsecondaries` can not be used with the `-a` option to add a host into an existing device group.

[`failback=`{`enabled` | `disabled`}]
The `failback` behavior of a disk device group can be `enabled` or `disabled` with either the `add` or `change` forms of the command.

This option is used to specify the behavior of the system should a disk device group primary leave the cluster membership and later return.

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

If `failback` is `enabled`, the disk device group will become mastered by the original primary. If `failback` is `disabled`, the disk device group will continue to be mastered by the secondary.

By default, `failback` is `disabled`.

[*otheroptions*]
Other disk device group type dependent options may be available for use with either the `add` or `change` forms 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)).

`-h` *node_options*
Add or remove a node from the cluster configuration database. When used with the `add` form of `scconf`, both the 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 assigned to access the node over the cluster interconnect is initialized to `clusternode<`*nodeid*`>-priv`. See the `-p` option.

`scconf` cannot be used by itself to add a new node to the cluster. All that this form of the command is able to do is to update the configuration database itself. It does not arrange to copy the configuration database onto the new node or to create the necessary node identifier on the new node. `scinstall`(1M) is the preferred command to add a node to a cluster.

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.

See the administration procedures in the Sun Cluster AnswerBook documentation for the complete procedure to remove a cluster node.

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

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

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

`-h node=`*nodename*

You must specify the `node=`*node*option with any occurrence of the `-h` option.

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

`-h node=`*node*

`-m` *cable_options*
Help 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 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 which accept `-m` are as follows:

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

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

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

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

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

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

The following suboptions are supported:

`endpoint=`[*node*:]*name*[*@port*]
: An `endpoint` suboption must always 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, it 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`; and, 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. Please refer to the individual cluster transport adapter and cluster transport junction man pages for more information regarding *port* assignments and other requirements (e.g., `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*
: This suboption may be used to `change` the state of a cable and the two end points 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 `enabled`. 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 it is added to the configuration. But, to add a cable in the `disabled` state, `noenable` can be used as part of an add.

`noenable`
: The `noenable` suboption may be used when adding a cable to the configuration. By default, adding a cable causes the cable, the two ports to which it is connected, and the the adapters or junctions on which the ports are found to

all have their `states` set to enable. But, if `noenable` is given when adding 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.

`-P` *privatehostname_options*
Use this option with either the `add` or `change` forms of the command to specify a hostname 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 hostname is clusternode<*nodeid*>-priv.

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

Both the `add` and `change` forms of `scconf` behave identically with regards to `-P`. The `-P` *privatehostname_options* for each of the two forms of the command which accept `-P` are as follows:

```
add:    -P node=node[,privatehostname=hostalias]
change: -P node=node[,privatehostname=hostalias]
```

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

[`privatehostname=`*hostalias*]
This option supplies the host alias to be used for accessing a node over the private cluster interconnect, or transport. If no `privatehostname` option is given, the private hostname for the given `node` is reset to the default.

`-q` *quorum_options*
Use this option for managing shared cluster quorum devices and various cluster quorum properties. 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 cluster quorum configuration propterties or states. The `-q` *quorum_options* available for each of the three forms of the command which can be used to change the cluster quorum configuration are as follows:

```
add:    -q globaldev=devicename[,node=node,node=node[,...]]
change: -q node=node,{maintstate | reset}
change: -q globaldev=devicename,{maintstate | reset}
change: -q reset
change: -q installmode
remove: -q globaldev=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` option to reset the quorum information.

`globaldev=`*devicename*

This option is used to specify the name of a global disk device to use when adding or removing a shared quorum device to or from the cluster. It 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.

With the `add` form of the command, if a `globaldev` is given 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.

The `change` form of `scconf` can be used with `-q globaldev` 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 non-zero vote counts which have ports to the device.

`node=`*node*

This option can be used with the `add` form of the command to select which nodes should be configured with ports to the shared quorum device being added. It can also be used with the `change` form of the command to change the quorum state of a node.

When the `node` option 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` will return 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, it's vote count is reset to one 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* may be given either as a node name or node ID.

[`maintstate`]

This option may be used as a flag with the `change` form of the command, for either the `globaldev` or `node` options, in order to put 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 can be useful when a node or device must be shut down for an extended period in order to perform maintenance. Once a node boots back into the cluster, it will normally remove itself from maintenance mode.

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

`[reset]`
This option is used as a flag with the `change` form of the command in order to reset the configured quorum vote count of a shared quorum device or node. The option may be combined with either the `globaldev` or `node` options, or it may stand by itself.

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.

`installmode`
This option can be used to force the cluster back into `installmode`. 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, a quorum config `scconf reset` command should be issued to reset the vote counts to their defaults and to clear the `installmode` setting.

`-T` *authentication_options*
Establish authentication policies for nodes 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; and, if so, the joining node is authenticated. The default is to allow any machine to add itself to the cluster (see `scinstall`(1M)).

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

```
add:    -T node=nodename[,...][,authtype=authtype]
change: -T authtype=authtype
remove: -T {node=nodename[,...] | all}
```

`node=`*nodename*
At least one `node` suboption is required for the `add` form of the command and is optional for `remove`. It is used to add or remove host names from the list of nodes that are able to install and configure themselves as nodes in the cluster. If the authentication list is empty, any host may 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 prevents any host from attempting to install and configure itself into the cluster.

The list may be cleared of all node names by giving `all` as the argument to `-T` with the `remove` form of the command. A cleared authentication list means that any node may attempt to install and configure itself into the cluster.

`authtype=`*authtype*
The `authtype` suboption may be given with either the `add` or `change` forms of the command.

The only currently supported `authtype`s are `des`, and `sys` (or, `unix`). The default authentication type is `sys`, offering 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.

`-v`
The `-v`, or verbose, option can be used with the `print` form of the command to request a more detailed listing of the cluster configuration. If used with other options, additional information may be printed when an error is encountered.

EXAMPLES

EXAMPLE 1 Typical Post-installation Setup Operations

The following commands provide an example of a typical set of post-installation setup operations which 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 which 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 \
    -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`	Successful completion.
`non-zero`	An error has occurred.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscu

SEE ALSO

`scconf_dg_sds`(1M),`scconf_dg_svm`(1M), `scconf_dg_vxvm`(1M), `scconf_dg_rawdisk`(1M), `scconf_transp_adap_hme`(1M), `scconf_transp_adap_eri`(1M), `scconf_transp_adap_sci`(1M), `scconf_transp_jct_etherswitch`(1M), `scconf_transp_jct_dolphinswitch`(1M), `hosts`(4), `nsswitch.conf`(4), `publickey`(4), `attributes`(5), `sctransp_rsm`(7P), `sctransp_dlpi`(7P)

NOTES

You should either backup the root filesystem on every node after changing the configuration using `scconf`, or keep a log of all changes so that if you need to recover configuration changes between normal system backups, you can use the log to return to the most recent configuration.

Option lists given with `scconf` command are always considered in the order given 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. It is always best to try to group all related options of this type together on a single command line, in order to reduce overhead to the cluster.

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
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.

Solstice DiskSuite has the concept of a multihosted or shared diskset, which is a grouping of two or more hosts and disk drives which are accessible by all hosts and 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 disks 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 currently 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 the `preferenced` suboption is not specified with the "change", the already established `true` or `false` setting is used.

The `scconf` command cannot be used to remove the Solstice DiskSuite device group from the cluster configuration, the Solstice DiskSuite `metaset` command is used to do this. 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.

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.

Solaris Volume Manager has the concept of a multihosted or shared diskset, which is a grouping of two or more hosts and disk drives which are accessible by all hosts and 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 disks 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 the `preferenced` suboption is not specified with the "change", the already established `true` or `false` setting is used.

The `scconf` command cannot be used to remove the Solaris Volume Manager device group from the cluster configuration, the Solaris Volume Manager `metaset` command is used to do this. 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/update VxVM device group configuration.

SYNOPSIS

scconf -a -D type=vxvm,generic_options

scconf -c -D generic_options, [sync]

scconf -r -D name=<device group name>

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. This involves defining a name for the new device group, specifying the nodes on which this group can be accessed, and specifying a set of properties used to control actions.

For VxVM disk device groups, only one VxVM disk group can be assigned to a disk device group, and the disk device group name always matches the name of the VxVM disk group. It is not possible to create a VxVM disk device group unless the corresponding VxVM disk group is first imported on one of the nodes in that device's nodelist.

Before a node can be added to a VxVM disk device group, every physical disk in the disk group must be physically ported to that node. After registering the disk group as a VxVM disk device group, it must first be deported from the current node owner and the auto-import flag must be turned off for the disk group.

To create a VxVM disk device group for a disk group, the `scconf`(1M) command must be run 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.

If you want to change the order of node preference list from `false` to `true` all the nodes currently existing in the device group must be specified in the `nodelist`. 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.

The `sync` option is used for synchronizing the clustering software with VxVM diskgroup volume information. `sync` is only valid with the change form or the command, and it should be used whenever a volume is added to or removed from a device group.

The remove (`-r`) option removes a VxVM device group from the Sun Cluster device groups configuration. This form of command can also be used to remove the nodes from the VxVM disk device group configuration.

OPTIONS

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

The following action options 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 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`(1M) 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.

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

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

SEE ALSO

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

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`.

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_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_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 adapters can be configured as cluster transport adapters. These adapters can be used with transport types dlpi and rsm.

An SCI adapter can only be connected to another SCI adapter or to a Dolphin SCI switch. When an SCI adapter is connected to a Dolphin 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 scconf(1M) or scinstall(1M). 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 scconf or scinstall 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), 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.

When a transport junction is used and the endpoints of the transport cable are configured using `scconf`, `scinstall`, 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), `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

scdidadm – disk ID 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*

/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 disk ID (DID) pseudo device driver.

There are four primary operations that it performs: creating driver configuration files, modifying entries in the file, loading the current configuration into the kernel and listing 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 the current device mapping between the disks present and the corresponding disk ID and DID instance number.

The `devfsadm`(1M) creates the filesystem device entry points.

OPTIONS

The following options are supported:

`-c`	Perform 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.
`-C`	Remove all DID references to underlying devices which have been detached from the current node. This option should be used after the Solaris device commands have been used to remove references to non-existent devices on the cluster nodes. You can only use this option from a node that is booted in cluster mode.
`-h`	Print a header when listing device mappings. This option is meaningful only when used with the `-l` and `-L` flags.
`-i`	Initialize the DID driver. This step is necessary to enable I/O requests to the DID driver.

`-l`	List the local devices in the DID configuration file. The output of this command can be customized using the `-o` flag. When no `-o` options are specified, the default listing displays the *instance* number, the local *fullpath* and the *fullname.*
`-L`	List 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` flag. When no `-o` options are specified, the default listing displays the *instance* number, all local and remote *fullpath* strings and the *fullname.*
`-o` *fmt*	List the devices currently known to the did driver according to the format specification fmt. Multiple `-o` options can be specified; the fmt specification will be interpreted as a comma separated list of format option-arguments. This option is meaningful only when used with the `-l` and `-L` flags. The available format option-arguments include:

instance	Print the instance number of the device known by the DID driver, for example, *1*.
path	Print the physical path name of the device associated with this device ID, for example, */dev/rdsk/c0t3d0*.
fullpath	Print the full physical path name of the device associated with this device ID. This includes the host where the path is located. For example, *phys-hostA:/dev/rdsk/c0t3d0*.
host	With the `-L` option, print the names of all hosts that have connectivity to the specified device, one per line. With the `-l` option, print the name of the local host that has connectivity to the specified device.
name	Print the DID name of the device associated with this device ID for example, *d1*.
fullname	Print the full DID path name of the device associated with this device ID, for example, */dev/did/rdsk/d1*.
diskid	Print the hexadecimal representation of the disk ID associated with the instance of the device being listed.
asciidiskid	Print the ASCII representation of the disk ID associated with the instance of the device being listed.

`-r`	Reconfigure the database. This flag results in a thorough search of the rdsk and rmt device trees. A new instance number is assigned for all device IDs not seen before. A new path is added for each newly discovered device. You can only use this option from a node that is booted in cluster mode.

`-R` *device* Perform repair procedures for a particular device instance. The argument to this command can be a particular physical device path that has been replaced with a new disk, or the instance number of the device that was just replaced.

This command does not modify the driver behavior until each of the nodes physically attached to the device has been rebooted.

This option also ensures correct SCSI reservation state on the specified device.

You can only use this option from a node that is booted in cluster mode.

`-U` Convert an existing `/etc/did.conf` file into a set of Cluster Configuration Repository (CCR) tables. If the tables already exist, this command will fail.

`-u` Load the device ID 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.

`-v` Print the version number of this program.

EXAMPLES

EXAMPLE 1 Adding Devices Attached to the Local Host to the CCR

The following example adds 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 corresponding 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:

```
/dev/dsk/c1t4d0 d1
```

EXAMPLE 4 Performing a Repair Procedure

The following example performs the repair procedures for a particular device path. Assuming the disk /dev/dsk/c1t4d0 has been replaced with a new disk with a new disk ID. The database is updated to reflect this new disk ID as the one corresponding to the instance number previously associated with the old disk ID:

```
% 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 disk c1t4d0s0 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	Successful completion.
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), scgdevs(1M), attributes(5),did(7)

Sun Cluster System Administration Guide

NOTES

There is one caveat regarding multiported tape drives and CD-ROM drives. Each mulitported tape or CD-ROM drive appears in the namespace once per physical connection.

NAME

scgdevs – global device 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 it follows that each physical device is visible across the cluster. This means that any disk, tape or CD ROM 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 device namespace without requiring a system reboot. The `drvconfig`(1M) and `devlinks`(1M) commands must be executed prior to the 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 it is run from a node that is not a cluster member, it exits with an error code and leaves the system state unchanged.

EXIT STATUS

The following exit values are returned:

`0`	Successful completion.
`non-zero`	An error occurred. Error messages are displayed on the standard output.

FILES

`/devices`	Device nodes directory
`/global/.devices`	Global device 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
Interface Stability	Evolving

SEE ALSO

`boot`(1M), `devfsadm`(1M), `devlinks`(1M), `scdidadm`(1M), `attributes`(5), `did`(7)

Sun Cluster System Administration Guide

NOTES 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

scinstall -p [-v]

/usr/cluster/bin/scinstall -r [-N *cluster-member*] [-G *mount-point*]

cdrom-mnt-pt/**SunCluster_3.1/Sol**_*release*/**Tools/scinstall**

cdrom-mnt-pt/**SunCluster_3.1/Sol**_*release*/**Tools/scinstall** -i [-k] [-d *cdimage-dir*] [-s *srvc* [,…]] [-F [-C *clustername*] [-T *authentication-options*] [-G {*special* | *mount-point*}] [-A *adapter-options*] [-B *junction-options*] [-m *cable-options*] [-w *netaddr-options*]]

cdrom-mnt-pt/**SunCluster_3.1/Sol**_*release*/**Tools/scinstall** -i [-k] [-d *cdimage-dir*] [-s *srvc* [,…]] [-N *cluster-member* [-C *clustername*] [-G {*special* | *mount-point*}] [-A *adapter-options*] [-B *junction-options*] [-m *cable-options*]]

cdrom-mnt-pt/**SunCluster_3.1/Sol**_*release*/**Tools/scinstall** -a *install-dir* [-d *cdimage-dir*]

cdrom-mnt-pt/**SunCluster_3.1/Sol**_*release*/**Tools/scinstall** -c *jumpstart-dir* -h *nodename* [-d *cdimage-dir*] [-s *srvc* [,…]] [-F [-C *clustername*] [-G {*special* | *mount-point*}]] [-T *authentication-options* [-A *adapter-options*] [-B *junction-options*] [-m *cable-options*] [-w *netaddr-options*]]

cdrom-mnt-pt/**SunCluster_3.1/Sol**_*release*/**Tools/scinstall** -c *jumpstart-dir* -h *nodename* [-d *cdimage-dir*] [-s *srvc* [,…]] [-N *cluster-member* [-C *clustername*] [-G {*special* | *mount-point*}] [-A *adapter-options*] [-B *junction-options*] [-m *cable-options*]]

cdrom-mnt-pt/**SunCluster_3.1/Sol**_*release*/**Tools/scinstall** -u *upgrade-mode* [*upgrade-options*]

DESCRIPTION

The `scinstall` utility 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`).

  This form of the `scinstall` utility must always be run 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 CD-ROM 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 `setup_install_server`(1M)).
- 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 `add_install_client`(1M)).

- 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`, which has several modes and options, upgrades a Sun Cluster node. This form of the `scinstall` command must always be run 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, `scinstall` attempts to run in interactive mode.

All forms of the command other than the `print release` form (`-p`) must be run as superuser.

`scinstall` is located in the `Tools` directory on the Sun Cluster CD-ROM. If the Sun Cluster CD-ROM has been copied to a local disk, *cdrom-mnt-pt* is the path to the copied Sun Cluster CD-ROM image. The SUNWscu software package also includes a copy of `scinstall`.

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`

Specify the `set up install server` form of the `scinstall` command. It is used to create an *install-dir* on any Solaris machine from which the command is run and then copy a Sun Cluster CD-ROM to that directory.

If the *install-dir* already exists, `scinstall` 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 `setup_install_server`(1M)).

`-c`

Specify the `add install client` form of the `scinstall` command. This 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 *nodename* install client (see `add_install_client`(1M)).

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*. And, if not already installed, special JumpStart `class` files and `finish` scripts supporting cluster initialization are added to the *jumpstart-dir*. Configuration data used by the Sun Cluster-supplied finish script is established for each node set up using this method.

Users can customize the Solaris class file installed by using the `-c` option to `scinstall` by editing it 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* set up with the `-c` option as an automated Sun Cluster JumpStart install client, a configuration directory is set up 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. These files should never be edited directly.

You can customize the JumpStart configuration in the following ways.

- A user-written finish script can be added as the file name:

  jumpstart-dir`/autoscinstall.d/nodes/`*nodename*`/finish`

  User-written finish scripts are run after the finish script supplied with the product.
- If the directory

  jumpstart-dir`/autoscinstall.d/nodes/`*nodename*`/archive`

  exists, all files in that directory are copied to the new installation. In addition, if an `etc/inet/hosts` file exists in that directory, the hosts information found in that file is used 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, all files in that directory are installed by 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 created under *jumpstart-dir*.

See `add_install_client`(1M)) and related JumpStart documentation for more information on setting up custom JumpStart install clients.

This form of the command should be run from the *install-dir* (see the `-a` form of `scinstall`) on the JumpStart server used for initializing the cluster nodes.

Before using `scinstall` to set up a node as a custom Sun Cluster JumpStart client, each node must also first be established as a Solaris install client. The JumpStart directory used with the `-c` option to `add_install_client`(1M) should be the same as that used with the `-c` option to `scinstall`. However, the `scinstall` *jumpstart-dir* does not have a server component to it, since `scinstall` must be run from a Solaris JumpStart server.

Removing a node as a custom Sun Cluster JumpStart client simply involves removing it from the `rules` file.

`-i` Specify the `install` form of the `scinstall`command. It can both install Sun Cluster software and initialize a node as a new cluster member. The new node is the node from which `scinstall` is run.

If the `-F` option is used with `-i`, `scinstall` will establish the node as the first node in a new cluster.

If the `-N` option is used with `-i`, `scinstall` will add the node to an already-existing cluster.

If the `-s` option is given and the node is an already-established cluster member, only the indicated *srvc* (data service) is installed.

`-p` Print 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` which can be run as a non-root user.

`-r` Remove 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` *upgrade-mode*
Upgrade Sun Cluster software on the node executing the `scinstall` command. The upgrade form of `scinstall` will have several different modes of operation, depending upon the releases involved, as specified by *upgrade-mode*. See `Upgrade Options` below for information specific to the type of upgrade that you intend to perform.

Additional Options

Additional options can be combined with the basic options to modify the default behavior of each form of the command. Refer to the `SYNOPSIS` section for additional details on which of these options are legal with which forms of `scinstall` .

These following additional options are supported:

`-k`
This option is only legal with the `install` (`-i`) form of the command.

If this option is given, no attempt is made to install the Sun Cluster software packages. Without this option, the default is to install any cluster packages which are not already installed.

`-d` *cdimage-dir*
This option is legal with all forms of the command other than the interactive and `print release` (`-p`) forms.

This option is used to specify an alternate directory location for finding the CD-ROM images of the Sun Cluster product and unbundled Sun Cluster data services. If this option is not specified, the default is the parent of the product directory (that is, `/SunCluster_3.1`) from under which the current instance of `scinstall` was started.

`-F` [*config-options*]
The `-F` option can only be used with the `install` (`-i`), `upgrade` (`-u`), or `add install client` (`-c`) forms of the command.

The `-F` option is only used to establish the first node in the cluster. The installation of secondary nodes will block 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.

`-h` *nodename*
This option is used 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.

`-N` *cluster-member* [*config-options*]
The `-N` option can only be used 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 3.1 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. `scinstall` 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.

`-s` *srvc*[`,`...]
This option can only be used with the `install` (`-i`), `upgrade` (`-u`), or `add install client` (`-c`) forms of the command to install or upgrade given *srvc* (data service package).

If a data service package cannot be located, a warning message is printed, but installation otherwise continues to completion.

`-v`
This option can only be used 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.

Configuration Options

The *config-options* which can be used with the `-F` option or `-N` *cluster-member* option are as follows.

```
cdrom-mnt-pt/SunCluster_3.1/Sol_release/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] ]
        [-w netaddr-options ]
      ]
      [-N cluster-member
        [ -C clustername ]
        [ -G {special  | mount-point} ]
        [ -A adapter-options ]
        [ -B junction-options ]
        [ -m endpoint=[this-node]:name[@port],endpoint= \
           [node:]name[@port] ]
      ]
```

`-C` *clustername*
This option is only legal when the `-F` or `-N` option is also given.

It is used to specify the name of the cluster. If this 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 this is a node 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* which is not the name of the cluster to which *cluster-member* belongs.

`-G` {*special* | *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 to use. 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 `scinstall`(1M) is run, `scinstall`(1M) attempts to add the necessary entry to the `vfstab`(4) file when it does not find a `/global/.devices/node@`*nodeID* mount.

By default, `scinstall`(1M) looks for an empty file system mounted on `/globaldevices`. If such a file system is provided, `scinstall`(1M) makes the necessary changes to the `vfstab`(4) 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 `newfs`(1M). 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 100 Mbytes in size. If this partition or file system is not available, or is not large enough, it might be necessary to re-install 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*

This option is only legal when the `-F` option is also given.

This option is used to establish authentication policies for nodes attempting 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.

The `-T` option can only be used with `scinstall` when setting up the very first node in the cluster. If the authentication list or policy needs to be changed on an already-established cluster, `scconf`(1M) can be used.

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*[,...]
: At least one `node` suboption is required when `-T` is given. It is used to add host 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 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. This list of nodes can be modified or cleared at any time by using the `scconf`(1M) command from one of the active cluster nodes.

[`authtype=`*authtype*]
: The only currently-supported `authtypes` are `des` and `sys` (or, `unix`). If no `authtype` is given, `sys` is the default.

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

  The authentication type can be changed at any time by using the `scconf`(1M) command from one of the active cluster nodes.

`-A` *adapter-options*
: This option is only legal when the `-F` or `-N` option is also given. Each occurrence of the `-A` option is used to configure 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 transport type is `dlpi`, it is not necessary to use the `trtype` suboption. And so, either of the two following forms can be used for specifying the `-A` *adapter-options*.

  `-A [trtype=type,]name=`*adaptername*`[,`*other-options*`]`
  `-A` *adaptername*

  [trtype=*type*]
  : The `trtype` option can be used with each occurrence of `-A` and is used to specify a transport type for the adapter. Examples of transport `types` are `rsm` and `dlpi` (see `sctransp_rsm`(7P) and `sctransp_dlpi`(7P)).

    The default transport type is `dlpi`.

name=*adaptername*

An *adaptername* option must be given with each occurrence of `-A`. An *adaptername* is constructed from a *device name*, immediately followed by a *physical-unit* number (for instance, `hme`).

If no other suboptions are needed with `-A`, the *adaptername* can be given as a standalone argument to `-A` (that is, `-A` *adaptername*).

[*other-options*]

When other options are available for a particular adapter type, they can be used with `-A`. Refer to the individual cluster transport adapter man pages (for instance, `scconf_transp_adap_hme`(1M), `scconf_transp_adap_eri`(1M), and `scconf_transp_adap_qfe`(1M)) for information on special options which might be used with them.

`-B` *junction-options*

This option is only legal when the `-F` or `-N` option is also given. Each occurrence of the `-B` option is used to configure a cluster transport junction. Examples of such devices can include, but are not limited to, Ethernet hubs, other switches of various types, and rings.

If no `-B` options are given, an attempt is made add a default junction at the time that the first node is instantiated as a cluster node. When additional nodes are added to the cluster, additional junctions are not added by default; however, they can be added explicitly. The default junction is named `hub1`, and it is of type `switch`.

When the junction type is type `switch`, it is not necessary to use the `type` suboption. And so, either of the two following forms can be used for specifying the `-B` *junction-options*.

```
-B [type=type,]name=name[,other-options]
-B name
```

If a cluster transport junction is already configured for the given *name*, a message is printed and the `-B` option is ignored.

In the case of directly-cabled transport adapters, it might not be necessary to configure any junctions whatsoever. To avoid configuring default junctions, the following special `-B` option can be given.

```
-B type=direct
```

[type=*type*]

The `type` option can be used with each occurrence of `-B` and is used to specify a junction type. Ethernet hubs and SCI switches are examples of cluster transport junctions which are both `type switch` (see `scconf_transp_jct_dolphinswitch`(1M) and `scconf_transp_jct_etherswitch`(1M)).

The `type` can also be set to `direct`, in order to suppress the configuration of any default junctions. Junctions do not exist in a transport configuration made up of only directly-connected transport adapters. So, when `type` is set to `direct`, it is not necessary to include a `name` suboption.

name=*name*
Unless the `type` is `direct`, a `name` must be always be given with each occurrence of the `-B` option, in order to specify the name of a cluster transport junction. 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`, the junction *name* can be given as a standalone argument to `-B` (that is, `-B` *name*).

[*other-options*]
When other options are available for a particular junction type, they can be used with `-B`. Refer to the individual cluster transport junction man pages (for instance, `scconf_transp_jct_etherswitch`(1M)) and `scconf_transp_adap_sci`(1M) for information on any special options which might be used with them.

`-m` *cable-options*
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 `scinstall`(1M) 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 no `-m` options are given, an attempt is made to configure a default cable. However, if more than one transport adapter or junction is configured with a given instance of `scinstall`(1M), it is not possible 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]
```

Two `endpoint` options must always be given with each occurrence of the `-m` option. The *name* component of the option argument is used to specify the name of either a cluster transport adapter or a 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, it 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 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 (for instance, `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/or junctions at each of the two endpoints of the cable must also be configured (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. An example of adding a cable from `scinstall`(1M) follows.

```
-m endpoint=:hme1,endpoint=hub1
```

In this example, port 0 of the `hme1` transport adapter on this node (the node which is being installed by `scinstall`(1M)) is cabled to a port on transport junction `4hub1`; the port used on `hub1` defaults to the node number of this node.

`-w` *netaddr-options*

This option is only legal when the `-F` option is also given.

It is used to specify a private network address (`networks`(4)) and, optionally, `netmasks`(4)) for use on the private network. It should only be necessary 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*

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*]

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.

Upgrade Options

The `-u` *upgrade-mode* and the *upgrade-option*s are as follows.

The `-u update` is used to upgrade a cluster node to a later Sun Cluster software release.

The *upgrade-options* to `-u update` are as follows.

cdrom-mnt-pt`/SunCluster_3.1/Sol_`*release*`/Tools/scinstall -u update` [`-s` { *srvc*[`,`...] | `all` } [`-d` *cdimage-dir*] [`-O`] [`-S` { `interact` | `testaddr=`*testipaddr@adapter*[`,testaddr=`...]}]

`-s` 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`
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 address option.

`-O`
This option overrides the hardware validation.

`-S`
This option allows the user either to direct the command to prompt the user for the required IP Network Multipathing (IPMP) addresses or to supply a set of IPMP test addresses on the command line for the conversion of NAFO to IPMP groups. See "IP Network Multipathing (Overview)" in *System Administration Guide: IP Services* for additional information on IP Network Multipathing.

It is illegal to combine both the `interact` and the `testaddr` forms on the same command line.

Note – The `-S` option is only required when one or more of the NAFO adapters in `pnmconfig` is not already converted to use IPMP.

The suboptions of the `-S` option are:

`interact`
Prompts the user to supply one or more IP Network Multipathing test addresses individually.

`testaddr=`*testipaddr@adapter*
Allows 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 `/etc/inet/hosts`) that will be assigned as routable, no-failover and deprecated test address to the adapter. IP Network Multipathing uses test addresses to detect failures and repairs. See "Administering Multipathing Groups With Multiple Physical Interfaces" in *System Administration Guide: IP Services* for additional information on configuring test addresses.

adapter
The name of the NAFO network adapter to be added to an IP Network Multipathing group.

EXAMPLES

EXAMPLE 1 Installing and Initializing a Two-Node Cluster

The following sequence of commands installs and initializes a typical two-node cluster. Insert the Framework CD-ROM and issue the following commands:

```
node1# cd /cdrom/cdrom0/SunCluster_3.1/Sol_8/Tools
node1# ./scinstall -i -F
node2# cd /cdrom/cdrom0/SunCluster_3.1/Sol_8/Tools
node2# ./scinstall -i -N node1
```

EXAMPLE 2 Setting Up a Solaris Install Server

The following sequence of commands arranges to set up a Solaris install server to install and initialize a three-node SCI cluster. Insert the Framework CD-ROM and issue the following commands:

```
installserver# cd /cdrom/cdrom0/SunCluster_3.1/Sol_9/Tools
installserver# ./scinstall -a /export/sc3.1
installserver# cd /export/sc3.1/SunCluster_3.1/Sol_9/Tools
installserver# ./scinstall -c /export/jumpstart \
     -h node1 -F -A hme2
installserver# ./scinstall –c /export/jumpstart \
     -h node2 -N node1 -A hme2
installserver# ./scinstall –c /export/jumpstart \
     -h node3 -N node1 -A hme2
```

EXAMPLE 3 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.

Do the following on each cluster node.

Insert the Framework CD-ROM and issue the following commands:

```
ok> boot -x
# cd /cdrom/cdrom0/SunCluster_3.1/Sol_9/Tools
# ./scinstall -u update -S interact
# cd /
# eject /cdrom/cdrom0
```

Insert the Data Services CD-ROM and issue the following commands:

```
# /usr/cluster/bin/scinstall -u update -s all \
 -d /cdrom/cdrom0
# reboot
```

EXAMPLE 4 Uninstall 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 and renames the global devices mount point to the default name `/globaldevices` and performs cleanup:

EXAMPLE 4 Uninstall a Node *(Continued)*

```
ok> boot -x
node4# cd /
node4# /usr/cluster/bin/scinstall -r
```

EXIT STATUS

The following exit values are returned:

`0`	Successful completion.
`non-zero`	An error occurred.

FILES

cdrom-mnt-pt`/.cdtoc`

cdrom-mnt-pt`/SunCluster_3.1/Sol_`*release*`/Tools/defaults`

cdrom-mnt-pt`/SunCluster_3.1/Sol_`*release*`/Product/.clustertoc`

cdrom-mnt-pt`/SunCluster_3.1/Sol_`*release*`/Product/.packagetoc`

cdrom-mnt-pt`/SunCluster_3.1/Sol_`*release*`/Product/.order`

cdrom-mnt-pt`/`*srvc*`/Product/.clustertoc`

cdrom-mnt-pt`/`*srvc*`/Product/.packagetoc`

cdrom-mnt-pt`/`*srvc*`/Product/.order`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	Solaris CD-ROM, SUNWscu

SEE ALSO

`newfs`(1M), `setup_install_server`(1M), `add_install_client`(1M), `patchadd`(1M), `sccheck`(1M), `scconf`(1M), `scconf_transp_adap_ge`(1M), `scconf_transp_adap_hme`(1M), `scconf_transp_adap_eri`(1M), `scconf_transp_adap_qfe`(1M), `scconf_transp_adap_sci`(1M), `scconf_transp_jct_etherswitch`(1M), `scconf_transp_jct_dolphinswitch`(1M), `scconf_transp_adap_sci`(1M), `clustertoc`(4), `netmasks`(4), `networks`(4), `packagetoc`(4), `order`(4), `attributes`(5), `sctransp_rsm`(7P), `sctransp_dlpi`(7P)

Sun Cluster 3.1 Software Installation Guide

System Administration Guide: IP Services

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* `| \*`}

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* `[-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*

LogicalHostname Resource Commands

scrgadm `-a -L -g` *RG_name* `[-j` *resource_name*`]` `-l` *hostnamelist* `[-n` *netiflist*`]` `[-y` *property* `[...]]`

SharedAddress 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

`-a -t` *resource_type_name*

form of the command.

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 command form:

`-a -g` *RG_name*

There are two types of resource group: failover and scalable.

A failover resource group is online on only one node at a time. A failover resource group may contain resources of any type although scalable resources configured into a failover resource group run on only one node at a time.

A scalable resource group may be online on several nodes at once. It may contain only resources which support scaling and may not contain resources which are constrained by their resource type definition only to failover behavior.

To create a failover resource group named MyDatabaseRG, use:

```
# scrgadm -a -g MyDatabaseRG
```

To create a scalable resource group named MyWebServerRG, use:

```
# scrgadm -a -g MyWebServerRG \
     -y Maximum_primaries=<integer>\
     -y Desired_primaries=<integer>
```

A newly created resource group is in an `UNMANAGED` state. Use the `scswitch`(1M) command (after creating resources in the group) to put a resource group in a `MANAGED` state.

To create a `resource` of a given type in a resource group use the command form:

`-a -j` *resource_name* `-t` *resource_type_name* `-g` *RG_name*

Creating a `resource` causes the underlying `RGM` mechanism to take several actions. It 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` and brings the `resource` online if it is enabled and its resource group is online.

To remove a resource group, first remove all `resources` from the containing 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 it.

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`
  Add a new configuration.

  Use with:

  `-t` to add a resource type

`-g`	to create a resource group
`-j`	to create a resource

`-c`
Modify existing configuration. Only values of the specified properties are set. Other properties retain their current values.

Use with:

`-t`	to modify a resource type
`-g`	to modify a resource group
`-j`	to modify a resource

`-r`
Remove configuration.

Use with:

`-t`	to remove a resource type
`-g`	to remove a resource group
`-j`	to remove a resource

`-p`
Display existing configurationinformation.

Use with:

`-v`[v]	to display more verbose output
`-t` *resource_type_name*	to display specific resource type configuration information
`-g` *resource_group_name*	to display specific resource group configuration information
`-j` *resource_name*	to display specific resource configuration information.

You can add up to two `-v` flags.

If you do not specify any `-t`, `-g`, or `-j` flags, the default is to provide information on all resource types, resource groups, and resources currently configured on the cluster.

Multiple `-t`, `-g`, and `-j` are supported and can be combined with any combination of `-v` options.

Target Options

Target options identify the target object.

The following target options are supported:

`-g` *RGname*	Resource group.
`-t` *resource_type_name*	Resource type.

`-j` *resource_name*	Resource. When used with `-a`, `-t` and `-g` must also be specified in the command to indicate the type of the resource to be instantiated and the containing resource group.

Resource Type Specific Options

The following resource type specific options are supported:

`-f` *registration_file_path*
This option is valid with `-a`. It is the pathname 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*
This option is valid with `-a` and `-c`. It is 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.

`-h` is required with the `-c` option. In this case, `-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. Comma is not allowed in a node name.

`-t` *resource_type_name*
This option 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. The registration filename is expected to be the same as the *resource_type_name*, and is expected to be present in the well-known directory where registration files are usually installed (`/usr/cluster/lib/rgm/rtreg`). Making the filename the same as the *resource_type_name* enables the shorthand notation:

```
# scrgadm -a -t SUNW.rt:2.0
```

instead of having to use the following notation:

```
# scrgadm -a -t stuff -f <full path to SUNW.rt:2.0>
```

To view the names of the currently registered resource types, use:

```
scrgadm -p
```

Starting in Sun Cluster 3.1, a resource type name is of the form

```
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, the recommended approach is to use the stock symbol for the company creating the resource type. The *resource_type_name* used with the `-t` flag may either be the full resource type name, or an abbreviation that leaves off 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 will fail.

The `scrgadm` command will fail the registration of the resource if the *RT_version* string includes a blank, tab, slash, backslash, asterisk, question mark, left square bracket, or right square bracket character,

You can omit the version component when specifying the *resource_type_name* used with the `-t` flag if there is not more than one version registered.

Resource type names created prior to Sun Cluster 3.1 continue to be of the form:

```
vendor_id.resource_type
```

Resource Group Specific Options

The following resource group specific options are supported:

`-h` *nodelist*
A shortcut for `-y Nodelist=`*nodelist*

`-y` *property*
This option is valid with `-a` and `-c`. *Property* is defined as *name*=*value* pair. Multiple instances of `-y`*property* are allowed.

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 case-insensitive.

See the `rg_properties`(5) man page for a description of the resource group properties.

Resource Specific Options

`-y` *property*
This option is valid with `-a` and `-c`. *Property* is defined as *name*=*value* pair. Multiple instances of `-y`*property* are allowed.

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 case-insensitive.

See the `rg_properties`(5) man page for a description of the resource properties.

`-x` *extension_property*
This option is valid with `-a` and `-c`. An *extension_property* is defined as *name*=*value* pair applicable only to a given `resource type`. Multiple instances of

`-x`*extension_property* are allowed.

To set a property to an empty value, use this option without specifying a value, as follows:

`-x` *property=*

For information on the extension properties available for a particular data service, see the man page for that data service.

Logical Hostname Specific Options

The following Logical Hostname specific options are supported:

These options apply to LogicalHostname resources. There are no special commands for removing a LogicalHostname resource. Use

`-r -j` *resource_name*

resource_name is the same name supplied with the optional `-j` flag when creating the LogicalHostname resource. If the `-j` flag and *resource_name* were omitted at LogicalHostname resource creation time then the name was generated by `scrgadm`. See the section describing `-j` under LogicalHostname specific options.

`-l` *hostnamelist*
The *hostnamelist* is a comma separated list of hostnames. It is the list of hostnames to be made available by this logical hostname resource. All hostnames in the same *hostnamelist* must be on the same subnet.

`-j` *resource_name* (optional)
Use this with with `-a` to explicitly name a logical hostname resource at creation and with `-r` to remove a resource from a resource group. If you do not use the `-j` option to explicitly name the `resource`, one is created whose name is the that of the first hostname in *hostnamelist*.

`-n` *netiflist* (optional)
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 (IPMP) group name, such as `sc_ipmp`. The *node* may be a node name or node ID. All nodes in the *nodelist* of the resource group must be represented in the *netiflist*. If `-n` *netiflist* is omitted, then an attempt is made to discover a net adapter on the subnet identified by the *hostnamelist* for each node in the *nodelist*. Single adapter IPMP groups are created for discovered network adapters not already in a IPMP group. Similarly, a single adapter IPMP group is created for a named adapter, if one does not already exist.

Refer to the `NOTES` section for more information.

`-y` *property*

Refer to the `Resource Specific Options` section for details.

Shared Address Specific Options

All `LogicalHostname` specific options apply to `SharedAddress` with the following additions:

`-l` *hostnamelist*
hostnamelist specifies the addresses to be shared. Dotted IP addresses may be specified although hostnames are strongly preferred.

`-X` *auxnodelist*
auxnodelist is a comma separated list of node names or IDs. Entries on this list must be members of the cluster. These are nodes which may host the specified `SharedAddresses`, but never serve as primary in the case of failover. This list is mutually exclusive with *nodelist*.

See the *nodelist* under `Resource Group Specific Options`.

EXIT STATUS

The following exit values are returned:

`0`	Successful completion. Can write a warning message to standard error when it completes successfully.
`non-0`	An error has occurred. Writes an error message to standard error when it exits with non-zero status.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

SEE ALSO

`ifconfig`(1M), `scswitch`(1M), `scstat`(1M), `rg_properties`(5), `attributes`(5)

NOTES

A network adapter that is not already configured for use cannot be discovered or placed into a IPMP group during `LogicalHostname` and `SharedAddress add` operations. See `ifconfig`(1M).

If `scrgadm` exits non-zero with the error message, `cluster is reconfiguring`, it is possible, though not definite, that the requested operation completed successfully, despite the error status. If the result is in doubt, 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 `scsetup`can 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, and a final message asking for confirmation.

The scshutdown command should only be run 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 of the transitions fail, scshutdown does not complete and displays an error message.
- Runs /usr/sbin/init 0 on all nodes. See init(1M).

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	A string that is sent out following the standard warning message: The system will be shut down in … 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

In the following example, scshutdown shuts down a cluster:

```
phys-palindrome-1# scshutdown
```

EXIT STATUS

The following exit values are returned:

0	Successful completion.
non-zero	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

SEE ALSO

`shutdown`(1M), `init`(1M), `attributes`(5)

NAME

scstat – monitor the status of Sun Cluster

SYNOPSIS

`scstat` `[-DWginpv [v] q] [-h` *node*`]`

DESCRIPTION

The `scstat` utility displays the current state of Sun Cluster and its components. Only one instance of the `scstat` utility 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 IPMP group and public network adapter

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 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 non-zero or times out, then the state of the resource will be `stop_failed`.

Possible resource states are: `online`, `offline`, `start_failed`, `stop_failed`, `monitor_failed`, or `online_not_monitored`.

Possible resource group states are: `unmanaged`, `online`, `offline`, `pending_online`, `pending_offline`, or `error_stop_failed`.

In addition to resource state, the RGM also maintains a resource status that can be set by the resource itself 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 printed after the status keyword.

The following are possible values for resource status and their descriptions:

`Online`	The resource is online and providing service.
`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.
`Unknown`	The current status is unknown or is in transition.
`Offline`	The resource is offline.

Device Groups

Device group status reflects on the availability of the devices in that group.

The following are possible values for device group status and their descriptions:

`Online`	The device group is online. There is a primary node and devices within the group are ready for I/O.
`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.
`Wait`	The device group is between statuses. This status might occur, for example, when a device goup is going from offline to online.
`Offline`	The device group is offline. There is no primary node. The device group must be brought online before any of its device can be used.

OPTIONS

You can specify command options to request the status for specific components.

If more than one option is specified, the `scstat` utility prints out the status in the specified order.

The following options supported:

`-D`	Show status for all disk device groups.
`-g`	Show status for all resource groups.
`-h` *node*	Show status for the specified node (*node*) and status of the disk device groups of which *node* is the primary node. Also show status of the quorum devices to which this node holds reservation, of the resource groups to which *node* is a potential master, and of the transport paths to which *node* is attached.
`-i`	Show status for all IPMP groups and public network adapters.
`-n`	Show status for all nodes.
`-p`	Show status for all components in the cluster. Use with `-v`[v] to display more verbose output.
`-q`	Show status for all device quorums and node quorums.
`-v`[v]	Show verbose output.
`-W`	Show status for cluster transport path.

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
```

EXAMPLE 1 Using the scstat Command *(Continued)*

These results are the same results you would see by typing the two commands:

```
% scstat -g
```

and

```
% scstat -h host
```

This is because, if more than one option is specified, `scstat` prints out the status in the specified order.

EXIT STATUS

The following exit values are returned:

`0`	Successful completion.
`non-zero`	An error has occurred.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

SEE ALSO

`scha_resource_setstatus`(1HA), `scha_resource_setstatus`(HA), `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 may 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 -S -h *from-node*

scswitch -R -h *node[,...]* -g *resource-grp[,...]*

scswitch {-u | -o}-g *resource-grp[,...]*

scswitch -z -g *resource-grp[,...]* -h *node[,...]*

scswitch -z -D *device-group[,...]* -h *node*

scswitch -Z [-g *resource-grp*]

DESCRIPTION

The `scswitch` utility moves resource groups or disk device groups to new primary nodes. It also evacuates all resource groups and disk device groups from a node by moving ownership elsewhere, brings resource groups or disk device groups offline and online, enables or disables resources, switches resource groups to or from an `unmanaged` state, or to clears error flags on resource groups.

You can run the the `scswitch` utility from any node in a Sun Cluster configuration. If a device group is offline, it can be brought online by a `scswitch` utility 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 nine forms fo the `scswitch` utility, 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` | `-n`)
Enable or disable the specified *resources*

`set maintenance mode` (`-m`)
Take the specified disk device groups offline from the cluster for maintenance. The resulting state survives reboots. If the disk device group is currently being accessed, then this action fails. 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.

`manage` (`-o`)
: Take the specified unmanaged *resource-grp*s out of the unmanaged state. This option brings the resource group under Resource Group Manager (RGM) management so that the RGM attempts to bring the resource group online.

`evacuate` or `switch all` (`-S`)
: Attempt 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. It is not necessarily the case that all evacuated groups are 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 by issuing an error message and exiting with a non-zero exit code

`restart` (`-R`)
: Take the specified resource groups offline and then back online on the specified primary *node*s of the resource groups.

`unmanage` (`-u`)
: Take the specified *resource-grp*s to the unmanaged state.

`set primaries` (`-z`)
: Cause the orderly transfer of one or more resource groups or disk device groups from one primary node in a Sun Cluster configuration to another node in the configuration (or to multiple nodes for resource groups configured with multiple primaries). It is also used to take resource groups offline and to bring disk device groups back online after being in maintenance.

`bring online` (`-Z`)
: Bring the specified *resource-grp*s online on the default list of primaries.

OPTIONS

The nine forms of the scswitch command are specified by the following options:

`-c`
: Clear 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 `SOFT` or `NONE`, 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 (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 3.0 Data Services Installation Guide*.

Make sure that both the resource and its monitor are stopped on the specified *node* before you clear the `STOP_FAILED` flag. If necessary, execute a `kill`(1) command on the associated processes.

`-e|-n` Enable" (`-e`) or disable (`-n`) the specified *resource*s.

If any resource dependencies exist in the resource group, then the resource cannot be disabled unless all resources which depend on it have been disabled. Conversely, a resource cannot be enabled until all of the resources on which it depends have been 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` Take 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 deported 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` Specify the `set maintenance mode` form of the `scswitch` utility.

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.

`-R` Specify the `restart` form of the command. The `-R` option moves the resource groups 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` Specify the `evacuate` or `switch all` form of the `scswitch` utility.

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|-o` Specify the `change RG state` form of the `scswitch` utility.

The `-u` option takes the specified managed *resource-grp*s to the unmanaged state. As a precondition of the `-u` option, all resources belonging 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` Specify a change in mastery of a resource group or a disk device group.

When used with the `-g` option, 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, "", then 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 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` Enable all resources of the specified *resource-grp* and their monitors, move the resource group into the managed state, and bring the resource group online on all the default primaries. When the `-g` option is not specified, `scswitch` utility attempts to bring all resource groups online.

The following options can combined with the previous nine options. Refer to the `SYNOPSIS` section for additional details about which of these options are legal with which forms of the `scswitch` utility.

`-D` Specify the name of one or more *device-group*s.

This option is only legal with the `-z`, `-F`, and `-m` options.

`-f` Use with the `-c` option to specify the error *flag-name*.

The only error flag currently supported is `STOP_FAILED.`

`-g` Specify the name of one or more *resource-grp*s.

This option is only legal with the `-z`, `-F`, `-R`, `-u`, `-o`, and `-Z` options.

`-h` Specify the names of one or more cluster *node*s.

This option is only legal with the `-z`, `-S`, `-R`, and `-c` options.

When used with the `-z`, `-R`, or `-c` 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 *resource-grp* or *device-group*, then the resource group or disk device group is not switched over.

`-j` Specify the names of one or more *resource*s.

This option is only legal with the `-e`, `-n`, and `-c` options.

`-M` Enable (`-e`) or disables (`-n`) monitoring for the specified *resource*s. 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 only legal with the `-e` and `-n` options.

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 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 3 Switching Over All Resource Groups and Disk Device Groups

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 4 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 5 Disabling Some Resources

The following command disables some resources:

```
node1# scswitch -n -j resource-1,resource-2
```

EXAMPLE 6 Enabling a Resource

The following command enables a resource:

```
node1# scswitch -e -j resource-1
```

EXAMPLE 7 Taking Resource Groups to the Unmanaged State

The following command takes resource groups to the unmanaged state:

```
node1# scswitch -u -g resource-grp-1,resource-grp-2
```

EXAMPLE 8 Taking Resource Groups to the Offline State

The following command take resource groups to the offline state:

```
node1# scswitch -o -g resource-grp-1,resource-grp-2
```

EXAMPLE 9 Switching Over a Device Group

The follwoing command switches over *device-group-1* to be mastered by *node2*.

```
node1# scswitch -z -h node2 -D device-group-1
```

EXAMPLE 10 Putting into Maintenance Mode

The following command puts *device-group-1* into maintenance mode:

```
node1# scswitch -m -D device-group-1
```

EXIT STATUS

This command blocks until its intended work is completely attempted.

The following exit values are returned:

`0`	Successful completion.
`non-zero`	An error has occurred. Writes an error message to standard output.

If `scswitch` exits `non-zero` with the error message `cluster is reconfiguring`, it is possible that the requested operation completed successfully, despite the error status. If the result is in doubt, you can execute `scswitch` again with the same arguments after the reconfiguration is complete.

If you invoke the `scswitch` utility 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.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Evolving

SEE ALSO

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

Sun Cluster 3.0 Data Services Installation Guide

NOTES

If you take resource groups 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 it offline. You can avoid this by first disabling the resources of the group, or by setting the `Desired_primaries` property of the group to zero.

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 `scvxinstall`:

- 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.

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 verifies that all other cluster nodes are running in cluster mode.
2. Disables Dynamic Multipathing (DMP), which is not supported with Sun Cluster software. DMP is disabled by creating symbolic links for the files `/dev/vx/dmp` and `/dev/vx/rdmp`.
3. Adds the `VRTSvxvm`, `VRTSvmdev`, and `VRTSvmman` packages for VxVM. If VxVM 3.2 or later is installed, the `VRTSlic` package is also installed.
4. 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.
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 `install only` mode *except* Step 5, then continues with the following additional tasks.

1. Installs the VxVM license key by running the `vxlicense` command.

2. Runs several VxVM commands to prepare for root-disk encapsulation.
3. Modifies the global devices entry in `/etc/vfstab` specified for the `/global/.devices/node@`*n* file system, where *n* is the node 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.
4. 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.
5. Unmounts the global devices file system by executing the startup file `/etc/rc2.d/S74scvxinstall`. The file system is automatically remounted after the encapsulation process is complete.
6. Recreates the special files for the `rootdg` volumes with a unique minor number on each node.

OPTIONS

The following options are supported:

`-d` *cdrom-image*	Specify the path to the VxVM packages.
`-e`	This option specifies the "install and encapsulate" mode of the `scvxinstall` utility. It installs VxVM and also encapsulates the root disk. 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`	This option specifies the "help" mode of the `scvxinstall` utility. It displays a brief help message about the `scvxinstall` utility.
`-i`	Specify the `install only` mode of the `scvxinstall` utility. It installs VxVM but does not encapsulate the root disk.
`-L` *license*	Specify a license key for the VxVM software. You can specify the `-L` *license* option multiple times to supply multiple license keys to `scvxinstall`. If you have no additional license keys to install, you can specify the word `none` for the *license* argument to `-L`.
`-s`	This option specifies the `show status` mode of the `scvxinstall` utility. It 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 packages but does not encapsulate the root disk. This example assumes that the VxVM CD-ROM is in the CD-ROM drive.

```
example#  scvxinstall -i
```

EXAMPLE 3 Installing the VxVM Packages Without Encapsulating the Root Disk

The following command installs the VxVM 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 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 packages and encapsulates the root disk. The command supplies the path to the CD-ROM images and the VxVM license key.

```
example#  scvxinstall -e -d /net/myserver/VxVM/pkgs \
             -L "9999 9999 9999 9999 9999 999"
```

EXIT STATUS

The following exit values are returned:

`0`	Successful completion.
`non-zero`	An error has occurred.

FILES

`/etc/rc2.d/S74scvxinstall.sh`
An `rc` script used to complete processing following a root-disk-encapsulation reboot

`/var/cluster/scvxinstall/*`
Location of temporary files used by `scvxinstall`

`/var/cluster/logs/install/scvxinstall.log.`*pid*
Log file created by `scvxinstall`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu, SUNWscr

SEE ALSO `scconf`(1M), `scinstall`(1M), `scsetup`(1M), `attributes`(5)

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/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_error_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/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/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 the 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 it 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 on 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 it 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_MAX_THRESOLD`, no action is taken. In addition, if the `probe_status` value is `0`, which indicates a successful health 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 it 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 health of the data service. A value of `0` implies that the recent data service health check was successful. A value of `SCDS_PROBE_COMPLETE_FAILURE` means complete failure and implies that the service is unhealthy. 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 health check. This value is reserved for future use.

RETURN VALUES

The `scha_cluster_open()` function returns the following:

`0`	The function succeeded.
`non-zero`	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/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), `scha_calls`(3HA), `scha_control`(3HA), `scha_fm_print_probes`(3HA), `scds_fm_sleep`(3HA), `scha_resource_setstatus`(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/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/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).

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize`(3HA).
sock	A handle to the socket established by this function. This is an output argument set by this function.
hostname	Name of the host where the process is listening.
port	TCP port number.
timeout	Timeout value in seconds.

RETURN VALUES

The `scds_fm_tcp_connect()` 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/libdsdev.h`
Include file

`/usr/cluster/lib/libdsdev.so`
Lbrary

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWscdev
Interface Stability	Evolving

SEE ALSO `scha_calls`(3HA), `scds_fm_tcp_disconnect`(3HA), `scds_get_rg_hostnames`(3HA), `scds_get_rs_hostnames`(3HA), `scds_initialize`(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_connect()` function terminates a TCP connection with a process being monitored.

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)
timeout	Timeout value in seconds

RETURN VALUES

The `scha_cluster_open()` 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/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_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/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/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/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/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/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/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/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-protocol combination 3-tuples in use by the resource. This is computed by combining the `Port_list` property settings on the resource in conjunction with all the settings on the resource in conjunction with all the hostnames in use by the resource as returned by the `scds_get_rs_hostnames()` function.

If the `Port_list` property for the resource specifies a hostname/port/for the resource specifies a hostname/port/protocol 3-tuple, this 3-tuple is returned by `scds_get_netaddr_list()`. If the hostname is omitted for a port/protocol combination, a hostname-port-protocol 3-tuple is computed for each hostname returned by `scds_get_rs_hostnames()` and the specified port/protocol combination specified.

Use `scds_get_netaddr_list()` in a fault monitor to compute the list of hostname-port-protocol 3-tuples in use by the resource and to monitor the resource for all such 3-tuples.

Values for the protocol type are defined in header file `<netinet/in.h>`.

Free the memory allocated and returned by this function with `scds_free_netaddr_list()`.

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize()`
netaddr_list	List of hostname-port-protocol 3-tuples used by the resource group

RETURN VALUES

The `scds_get_netaddr_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/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/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/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/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/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/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/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), `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/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/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/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/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/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/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/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>`

`void`**`scds_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/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/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()`.

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[]*), then this 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:

Property to Retrieve
   Function Name

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>*

The recognition of property names is case insensitive.

Resource Specific Functions

The function declaration return 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.

Resource Property to Retrieve
  Function Declaration

```
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)

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_weak
  const scha_str_array_t *
  scds_get_rs_resource_dependencies_weak(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)

Start_timeout
  int scds_get_rs_start_timeout(scds_handle_t handle)

Stop_timeout
  int scds_get_rs_stop_timeout(scds_handle_t handle)
```

`Scalable`
  `boolean scds_get_rs_scalable(scds_handle_t handle)`

`Thorough_probe_interval`
  `scds_get_ext_`<*property-name*>

Resource Group Specific Functions

The function declaration return values for the resource group property to Rretrieve. 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.

Resource Group Property to Retrieve
  Function Declaration

`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)`

`Resource_dependencies_weak`
  `const scha_str_array_t * scds_get_rg_resource_dependencies_weak(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_mode`
  `scha_rgmode_t scds_get_rg_rg_mode(scds_handle_t handle)`

Resource Type Specific Functions

The function declaration return values for the Rresource 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.

Resource Type Property to Retrieve
  Function Declaration

```
API_version
  int scds_get_rt_api_version(scds_handle_t handle)

RT_basedir
  const char * scds_get_rt_global_rt_basedir(scds_handle_t
  handle)

Failover
  boolean_t scds_get_rt_implicit_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)

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)

RT_version
  const char * scds_get_rt_rt_version(scds_handle_t handle)
```

Extension Property Specific Functions

The function declaration return 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 in the table, 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()` function. You must use `scds_get_ext_property()` to retrieve extension properties other than these four.

Extension Property to Retrieve
  Function Declaration

```
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 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/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_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/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/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_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).

PARAMETERS

The following parameters are supported:

handle	The handle returned from `scds_initialize`(3HA)
hostname	Internet hostname of the machine to connect to
port	Port number to make the connection with
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.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Indicates the function succeeded.
`SCHA_ERR_TIMEOUT`	Indicates the function timed out.

See `scha_calls`(3HA) for a description of other error codes.

FILES

`/usr/cluster/include/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

`close`(2), `connect`(3SOCKET), `scds_get_rg_hostnames`(3HA), `scds_get_rs_hostnames`(3HA), `scds_initialize`(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/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/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/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/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-type*`, 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), and `scha_strerror`(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 and all have the same general signature in that they take a *handle-type* argument returned from a previous call to an "open" function that 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 in the variable argument list and the type of a final "out" argument through which the requested information is returned. Repeated "get" calls can be made using the same handle until a "close" call which invalidates the handle and frees any memory allocated for values 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 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 below.

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 the RETURN VALUES section.

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`
Enum type indicating an `On_Off_switch` or `Monitor_switch` resource property value

```
typedef enum scha_switch {
    SCHA_SWITCH_DISABLED = 0,
    SCHA_SWITCH_ENABLED
} scha_switch_t;
```

`scha_rsstate_t`
Enum type indicating 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_t;
```

`scha_rgstate_t`
An enum type indicating 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_t;
```

`scha_failover_mode_t`
An enum type indicating a value for the `Failover_Mode` resource property

```
typedef enum scha_failover_mode {
    SCHA_FOMODE_NONE = 0,
    SCHA_FOMODE_HARD,
    SCHA_FOMODE_SOFT
} scha_failover_mode_t;
```

`scha_initnodes_flag_t`
An enum type indicating 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 indicating 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;
    char    **str_array;
} scha_str_array_t;
```

`array_cnt`	Gives the number elements in the list
`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`
This is 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_OK = 0,
    SCHA_RSSTATUS_OFFLINE,
    SCHA_RSSTATUS_FAULTED,
    SCHA_RSSTATUS_DEGRADED,
    SCHA_RSSTATUS_UNKNOWN
} scha_rsstatus_t;
```

`status`	holds an enum value indicating the resource status as set by the resource monitor

`scha_extprop_value_t`
This is 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_CHECK	sanity-checks failed
17 SCHA_ERR_RSTATUS	bad resource status
18 SCHA_ERR_INTERNAL	internal error was encountered

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_cluster_get`(3HA), `scha_control`(3HA), `scha_resource_get`(3HA), `scha_resourcegroup_get`(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`(3HA), `scha_cluster_get`(3HA), and `scha_cluster_close`(3HA) functions are used together to get to information about a cluster.

`scha_cluster_open`(3HA) initializes cluster access and returns an access handle to be used by `scha_cluster_get`(3HA). The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_cluster_get`(3HA) accesses cluster information as indicated by the *tag* argument. The *handle* is a value returned from a prior call to `scha_cluster_open`(3HA). 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`(3HA) remains intact until `scha_cluster_close`(3HA) is called on the handle used for the `scha_cluster_get`(3HA).

`scha_cluster_close`(3HA) takes a `handle` argument returned from a previous call to `scha_cluster_get`(3HA). It invalidates the handle and frees memory allocated to return values to `scha_cluster_get`(3HA) 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.

The macros defined in `<scha_tags.h>` that may be used 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**`.

It returns the name of 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.

  It returns the name of the node indicated by the numeric identifier.

`SCHA_ALL_NODENAMES`
: The output argument type is `scha_str_array_t**`.

  It returns the names of all nodes in the cluster.

`SCHA_ALL_NODEIDS`
: The output argument type is `scha_uint_array_t**`.

  It returns numeric node identifiers of all the nodes in the cluster.

`SCHA_NODEID_LOCAL`
: The output argument type is `uint_t*`.

  It 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.

  It returns the numeric node identifier of the node indicated by the name.

`SCHA_PRIVATELINK_HOSTNAME_LOCAL`
: The output argument type is `char**`.

  It returns the hostname by which the node that the command is run on as addressed on the cluster interconnect.

`SCHA_PRIVATELINK_HOSTNAME_NODE`
: The output argument type is `char**`. An additional argument is of type `char *`. The macro requires an additional unflagged argument that is the name of a cluster node.

  It returns the hostname by which the named node is addressed on the cluster interconnect.

`SCHA_ALL_PRIVATELINK_HOSTNAMES`
: The output argument type is `scha_str_array_t**`.

  It returns the hostnames 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*`.

  It 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.

It 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*`.

It returns the number of the `syslog`(3C) facility that is used for the cluster log.

SCHA_ALL_RESOURCEGROUPS
The output argument type is `scha_str_array_t**`.

It returns the names of all the resource groups that are being managed on the cluster.

SCHA_ALL_RESOURCETYPES
The output argument is type `scha_str_array_t**`.

It returns the names of all the resource types that are registered on the cluster.

SCHA_CLUSTERNAME
The output argument is type `char**`.

It returns the name of the cluster.

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.

EXAMPLES

EXAMPLE 1 Using the `scha_cluster_get`(3HA) Function

The following example uses the `scha_cluster_get`(3HA) function to get the names of all cluster nodes and 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;
```

EXAMPLE 1 Using the `scha_cluster_get`(3HA) Function *(Continued)*

```
        const char              *str;

        err = scha_cluster_open(&handle);
        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\n",
                    all_nodenames->str_array[ix], str);
        }
}
```

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES See 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), `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`(3HA), `scha_cluster_get`(3HA), and `scha_cluster_close`(3HA) functions are used together to get to information about a cluster.

`scha_cluster_open`(3HA) initializes cluster access and returns an access handle to be used by `scha_cluster_get`(3HA). The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_cluster_get`(3HA) accesses cluster information as indicated by the *tag* argument. The *handle* is a value returned from a prior call to `scha_cluster_open`(3HA). 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`(3HA) remains intact until `scha_cluster_close`(3HA) is called on the handle used for the `scha_cluster_get`(3HA).

`scha_cluster_close`(3HA) takes a `handle` argument returned from a previous call to `scha_cluster_get`(3HA). It invalidates the handle and frees memory allocated to return values to `scha_cluster_get`(3HA) 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.

The macros defined in `<scha_tags.h>` that may be used 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**`.

  It returns the name of 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.

It returns the name of the node indicated by the numeric identifier.

`SCHA_ALL_NODENAMES`
The output argument type is `scha_str_array_t**`.

It returns the names of all nodes in the cluster.

`SCHA_ALL_NODEIDS`
The output argument type is `scha_uint_array_t**`.

It returns numeric node identifiers of all the nodes in the cluster.

`SCHA_NODEID_LOCAL`
The output argument type is `uint_t*`.

It 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.

It returns the numeric node identifier of the node indicated by the name.

`SCHA_PRIVATELINK_HOSTNAME_LOCAL`
The output argument type is `char**`.

It returns the hostname by which the node that the command is run on as addressed on the cluster interconnect.

`SCHA_PRIVATELINK_HOSTNAME_NODE`
The output argument type is `char**`. An additional argument is of type `char *`. The macro requires an additional unflagged argument that is the name of a cluster node.

It returns the hostname by which the named node is addressed on the cluster interconnect.

`SCHA_ALL_PRIVATELINK_HOSTNAMES`
The output argument type is `scha_str_array_t**`.

It returns the hostnames 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*`.

It 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.

It 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*.

It returns the number of the syslog(3C) facility that is used for the cluster log.

SCHA_ALL_RESOURCEGROUPS
The output argument type is scha_str_array_t**.

It returns the names of all the resource groups that are being managed on the cluster.

SCHA_ALL_RESOURCETYPES
The output argument is type scha_str_array_t**.

It returns the names of all the resource types that are registered on the cluster.

SCHA_CLUSTERNAME
The output argument is type char**.

It returns the name of the cluster.

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.

EXAMPLES **EXAMPLE 1** Using the scha_cluster_get(3HA) Function

The following example uses the scha_cluster_get(3HA) function to get the names of all cluster nodes and 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;
```

EXAMPLE 1 Using the `scha_cluster_get`(3HA) Function *(Continued)*

```
        const char              *str;

        err = scha_cluster_open(&handle);
        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\n",
                    all_nodenames->str_array[ix], str);
        }
}
```

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES See 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), `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`(3HA), `scha_cluster_get`(3HA), and `scha_cluster_close`(3HA) functions are used together to get to information about a cluster.

`scha_cluster_open`(3HA) initializes cluster access and returns an access handle to be used by `scha_cluster_get`(3HA). The *handle* argument is the address of a variable to hold the value returned from the function call.

`scha_cluster_get`(3HA) accesses cluster information as indicated by the *tag* argument. The *handle* is a value returned from a prior call to `scha_cluster_open`(3HA). 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`(3HA) remains intact until `scha_cluster_close`(3HA) is called on the handle used for the `scha_cluster_get`(3HA).

`scha_cluster_close`(3HA) takes a `handle` argument returned from a previous call to `scha_cluster_get`(3HA). It invalidates the handle and frees memory allocated to return values to `scha_cluster_get`(3HA) 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.

The macros defined in `<scha_tags.h>` that may be used 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**`.

  It returns the name of 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.

It returns the name of the node indicated by the numeric identifier.

`SCHA_ALL_NODENAMES`
The output argument type is `scha_str_array_t**`.

It returns the names of all nodes in the cluster.

`SCHA_ALL_NODEIDS`
The output argument type is `scha_uint_array_t**`.

It returns numeric node identifiers of all the nodes in the cluster.

`SCHA_NODEID_LOCAL`
The output argument type is `uint_t*`.

It 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.

It returns the numeric node identifier of the node indicated by the name.

`SCHA_PRIVATELINK_HOSTNAME_LOCAL`
The output argument type is `char**`.

It returns the hostname by which the node that the command is run on as addressed on the cluster interconnect.

`SCHA_PRIVATELINK_HOSTNAME_NODE`
The output argument type is `char**`. An additional argument is of type `char *`. The macro requires an additional unflagged argument that is the name of a cluster node.

It returns the hostname by which the named node is addressed on the cluster interconnect.

`SCHA_ALL_PRIVATELINK_HOSTNAMES`
The output argument type is `scha_str_array_t**`.

It returns the hostnames 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*`.

It 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.

It 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*`.

It returns the number of the `syslog`(3C) facility that is used for the cluster log.

`SCHA_ALL_RESOURCEGROUPS`
The output argument type is `scha_str_array_t**`.

It returns the names of all the resource groups that are being managed on the cluster.

`SCHA_ALL_RESOURCETYPES`
The output argument is type `scha_str_array_t**`.

It returns the names of all the resource types that are registered on the cluster.

`SCHA_CLUSTERNAME`
The output argument is type `char**`.

It returns the name of the cluster.

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.

EXAMPLES **EXAMPLE 1** Using the `scha_cluster_get`(3HA) Function

The following example uses the `scha_cluster_get`(3HA) function to get the names of all cluster nodes and 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;
```

EXAMPLE 1 Using the `scha_cluster_get`(3HA) Function *(Continued)*

```
        const char              *str;

        err = scha_cluster_open(&handle);
        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\n",
                    all_nodenames->str_array[ix], str);
        }
}
```

FILES

`/usr/cluster/include/scha.h`	include file
`/usr/cluster/lib/libscha.so`	library

ATTRIBUTES See 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), `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(3HA) 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 *tag* argument indicates whether the request is to restart or relocate the resource or group. It should be a string value defined by one of the following macros that is defined in <scha_tags.h>:

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.

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(3HA) to keep count of recent restart attempts.

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

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 will be reflected in subsequent scha_resource_get(3HA) 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_GIVEOVER
Request 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 may 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 an SCHA_GIVEOVER request on that node within the interval specified by the Pingpong_interval property. In addition, the resources in the group might contain MONITOR_CHECK callback methods that the RGM invokes in the event of a giveover request. These methods verify that a node is healthy enough to run the resource. For example, a MONITOR_CHECK method might verify that an essential configuration file is available on the node, and if it is not, veto the giveover request.

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.

SCHA_CHECK_GIVEOVER
Perform all the same validity checks that would be done for an SCHA_GIVEOVER of the resource group named by the *rgname* argument, but do not actually relocate the resource group.

The `SCHA_CHECK_RESTART` and `SCHA_CHECK_GIVEOVER` 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 will be 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`(3HA) request. If the named resource is not in the resource group the request will be 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`(3HA) might be stopped as a result of the resource group going offline and so might never receive the return status of a successful request.

RETURN VALUES

The `scha_control()` function returns the following:

`0`	The function succeeded.
`non-zero`	The function failed.

ERRORS

`SCHA_ERR_NOERR`	Function succeeded.
`SCHA_ERR_CHECKS`	Request rejected: 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, will 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)

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_err_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 on 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 that the resource is configured in. 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 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 information. No value is returned for the out parameter 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 a subsequent calls.

`scha_resource_close()` takes a *handle* argument 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

The macros naming 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)).

`SCHA_DESCRIPTION`
The output argument type is `char**`.

`SCHA_TYPE`
The output argument type is `char**`.

`SCHA_RESOURCE_PROJECT_NAME`
The output argument type is `char**`.

`SCHA_ON_OFF_SWITCH`
The output argument type is `scha_switch_t*`.

`SCHA_MONITORED_SWITCH`
The output argument type is `scha_switch_t*`.

`SCHA_RESOURCE_STATE`
The output argument type is `scha_rsstate_t*`.

`SCHA_CHEAP_PROBE_INTERVAL`
The output argument type is `int*`.

`SCHA_THOROUGH_PROBE_INTERVAL`
The output argument type is `int*`.

`SCHA_RETRY_COUNT`
The output argument type is `int*`.

`SCHA_RETRY_INTERVAL`
The output argument type is `int*`.

`SCHA_FAILOVER_MODE`
The output argument type is `scha_failover_mode_t*`.

`SCHA_RESOURCE_DEPENDENCIES`
The output argument type is `scha_str_array_t**`.

`SCHA_RESOURCE_DEPENDENCIES_WEAK`
The output argument type is `scha_str_array_t**`.

`SCHA_NETWORK_RESOURCES_USED`
The output argument type is `scha_str_array_t**`.

`SCHA_SCALABLE`
The output argument type is `boolean_t*`.

`SCHA_PORT_LIST`
The output argument type is `scha_str_array_t**`.

`SCHA_LOAD_BALANCING_POLICY`
The output argument type is `char**`.

SCHA_LOAD_BALANCING_WEIGHTS
: The output argument type is `scha_str_array_t**`.

SCHA_AFFINITY_TIMEOUT
: The output argument type is `int*`.

SCHA_WEAK_AFFINITY
: The output argument type is `boolean_t*`.

SCHA_UDP_AFFINITY
: The output argument type is `boolean_t*`.

SCHA_STATUS
: The output argument type is `scha_status_value_t**`.

SCHA_START_TIMEOUT
: The output argument type is `int*`.

SCHA_STOP_TIMEOUT
: The output argument type is `int*`.

SCHA_VALIDATE_TIMEOUT
: The output argument type is `int*`.

SCHA_UPDATE_TIMEOUT
: The output argument type is `int*`.

SCHA_INIT_TIMEOUT
: The output argument type is `int*`.

SCHA_FINI_TIMEOUT
: The output argument type is `int*`.

SCHA_BOOT_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_MONITOR_CHECK_TIMEOUT
: The output argument type is `int*`.

SCHA_PRENET_START_TIMEOUT
: The output argument type is `int*`.

SCHA_POSTNET_STOP_TIMEOUT
: The output argument type is `int*`.

SCHA_NUM_RG_RESTARTS
: The output argument type is `int*`.

SCHA_NUM_RESOURCE_RESTARTS
: The output argument type is `int*`.

SCHA_TYPE_VERSION
: The output argument type is char**.

SCHA_STATUS_NODE
: The output argument type is scha_status_value_t**. An additional argument type is char*. The additional argument names the node. It returns the status of the resource on that node.

SCHA_RESOURCE_STATE_NODE
: The output argument type is scha_rsstate_value_t*. An additional argument type is char *. The additional argument names the node. It returns the state of the resource on that node.

SCHA_EXTENSION
: The output argument type is scha_extprop_value_t**. An additional argument type is char*. The additional argument names an extension property of the resource. It returns a structure containing the type and value of the property.

SCHA_ALL_EXTENSIONS
: The output argument type is scha_str_array_t**. It returns the names of all extension properties of the resource.

SCHA_GROUP
: The output argument type is char**. It returns the name of the resource group that the resource is configured in.

The macros naming 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_RT_DESCRIPTION	The output argument type is char**.
SCHA_RT_BASEDIR	The output argument type is char**.
SCHA_SINGLE_INSTANCE	The output argument type is boolean_t*.
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_FAILOVER	The output argument type is boolean_t*.
SCHA_API_VERSION	The output argument type is int*.
SCHA_RT_VERSION	The output argument type is char**.
SCHA_PKGLIST	The output argument type is scha_str_array_t**.
SCHA_START	The output argument type is char**.
SCHA_STOP	The output argument type is char**.
SCHA_VALIDATE	The output argument type is char**.
SCHA_UPDATE	The output argument type is char**.

SCHA_INIT	The output argument type is char**.
SCHA_FINI	The output argument type is char**.
SCHA_BOOT	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_MONITOR_CHECK	The output argument type is char**.
SCHA_PRENET_START	The output argument type is char**.
SCHA_POSTNET_STOP	The output argument type is char**.
SCHA_IS_LOGICAL_HOSTNAME	The output argument type is boolean_t*.
SCHA_IS_SHARED_ADDRESS	The output argument type is boolean_t*.

RETURN VALUES

These functions 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_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;
    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);

    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,
```

EXAMPLE 1 Using the `scha_resource_get()` Function *(Continued)*

```
          retry_count_out);

    printf("The log level for resource %s is %d\n", resource_name,
          loglevel_out->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_err_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 on 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 that the resource is configured in. 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 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 information. No value is returned for the out parameter 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 a subsequent calls.

`scha_resource_close()` takes a *handle* argument 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

The macros naming 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)).

SCHA_DESCRIPTION
The output argument type is char**.

SCHA_TYPE
The output argument type is char**.

SCHA_RESOURCE_PROJECT_NAME
The output argument type is char**.

SCHA_ON_OFF_SWITCH
The output argument type is scha_switch_t*.

SCHA_MONITORED_SWITCH
The output argument type is scha_switch_t*.

SCHA_RESOURCE_STATE
The output argument type is scha_rsstate_t*.

SCHA_CHEAP_PROBE_INTERVAL
The output argument type is int*.

SCHA_THOROUGH_PROBE_INTERVAL
The output argument type is int*.

SCHA_RETRY_COUNT
The output argument type is int*.

SCHA_RETRY_INTERVAL
The output argument type is int*.

SCHA_FAILOVER_MODE
The output argument type is scha_failover_mode_t*.

SCHA_RESOURCE_DEPENDENCIES
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_DEPENDENCIES_WEAK
The output argument type is scha_str_array_t**.

SCHA_NETWORK_RESOURCES_USED
The output argument type is scha_str_array_t**.

SCHA_SCALABLE
The output argument type is boolean_t*.

SCHA_PORT_LIST
The output argument type is scha_str_array_t**.

SCHA_LOAD_BALANCING_POLICY
The output argument type is char**.

`SCHA_LOAD_BALANCING_WEIGHTS`
The output argument type is `scha_str_array_t**`.

`SCHA_AFFINITY_TIMEOUT`
The output argument type is `int*`.

`SCHA_WEAK_AFFINITY`
The output argument type is `boolean_t*`.

`SCHA_UDP_AFFINITY`
The output argument type is `boolean_t*`.

`SCHA_STATUS`
The output argument type is `scha_status_value_t**`.

`SCHA_START_TIMEOUT`
The output argument type is `int*`.

`SCHA_STOP_TIMEOUT`
The output argument type is `int*`.

`SCHA_VALIDATE_TIMEOUT`
The output argument type is `int*`.

`SCHA_UPDATE_TIMEOUT`
The output argument type is `int*`.

`SCHA_INIT_TIMEOUT`
The output argument type is `int*`.

`SCHA_FINI_TIMEOUT`
The output argument type is `int*`.

`SCHA_BOOT_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_MONITOR_CHECK_TIMEOUT`
The output argument type is `int*`.

`SCHA_PRENET_START_TIMEOUT`
The output argument type is `int*`.

`SCHA_POSTNET_STOP_TIMEOUT`
The output argument type is `int*`.

`SCHA_NUM_RG_RESTARTS`
The output argument type is `int*`.

`SCHA_NUM_RESOURCE_RESTARTS`
The output argument type is `int*`.

SCHA_TYPE_VERSION
The output argument type is `char**`.

SCHA_STATUS_NODE
The output argument type is `scha_status_value_t**`. An additional argument type is `char*`. The additional argument names the node. It returns the status of the resource on that node.

SCHA_RESOURCE_STATE_NODE
The output argument type is `scha_rsstate_value_t*`. An additional argument type is `char *`. The additional argument names the node. It returns the state of the resource on that node.

SCHA_EXTENSION
The output argument type is `scha_extprop_value_t**`. An additional argument type is `char*`. The additional argument names an extension property of the resource. It returns a structure containing the type and value of the property.

SCHA_ALL_EXTENSIONS
The output argument type is `scha_str_array_t**`. It returns the names of all extension properties of the resource.

SCHA_GROUP
The output argument type is `char**`. It returns the name of the resource group that the resource is configured in.

The macros naming 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_RT_DESCRIPTION	The output argument type is `char**`.
SCHA_RT_BASEDIR	The output argument type is `char**`.
SCHA_SINGLE_INSTANCE	The output argument type is `boolean_t*`.
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_FAILOVER	The output argument type is `boolean_t*`.
SCHA_API_VERSION	The output argument type is `int*`.
SCHA_RT_VERSION	The output argument type is `char**`.
SCHA_PKGLIST	The output argument type is `scha_str_array_t**`.
SCHA_START	The output argument type is `char**`.
SCHA_STOP	The output argument type is `char**`.
SCHA_VALIDATE	The output argument type is `char**`.
SCHA_UPDATE	The output argument type is `char**`.

SCHA_INIT	The output argument type is char**.
SCHA_FINI	The output argument type is char**.
SCHA_BOOT	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_MONITOR_CHECK	The output argument type is char**.
SCHA_PRENET_START	The output argument type is char**.
SCHA_POSTNET_STOP	The output argument type is char**.
SCHA_IS_LOGICAL_HOSTNAME	The output argument type is boolean_t*.
SCHA_IS_SHARED_ADDRESS	The output argument type is boolean_t*.

RETURN VALUES

These functions 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_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;
    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);

    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,
```

EXAMPLE 1 Using the `scha_resource_get()` Function *(Continued)*

```
          retry_count_out);

    printf("The log level for resource %s is %d\n", resource_name,
          loglevel_out->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 on 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_RG_DESCRIPTION`
The output argument type is `char**`.

`SCHA_NODELIST`
The output argument type is `scha_str_array_t**`.

`SCHA_MAXIMUM_PRIMARIES`
The output argument type is `int*`.

`SCHA_DESIRED_PRIMARIES`
The output argument type is `int*`.

`SCHA_FAILBACK`
The output argument type is `boolean_t*`.

`SCHA_RESOURCE_LIST`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_STATE`
The output argument type is `scha_rgstate_t*`.

`SCHA_RG_DEPENDENCIES`
The output argument type is `scha_str_array_t**`.

`SCHA_GLOBAL_RESOURCES_USED`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_MODE`
The output argument type is `rgm_rgmode_t*`.

`SCHA_IMPLICIT_NETWORK_DEPENDENCIES`
The output argument type is `boolean_t*`.

`PINGPONG_INTERVAL`
The output argument type is `int*`.

`SCHA_PATHPREFIX`
The output argument type is `char**`.

`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. It returns the state of the resource group on that node.

`SCHA_RG_PROJECT_NAME`
The output argument type is `char**`.

RETURN VALUES

These functions return 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_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\n", 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 on 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_RG_DESCRIPTION`
The output argument type is `char**`.

`SCHA_NODELIST`
The output argument type is `scha_str_array_t**`.

`SCHA_MAXIMUM_PRIMARIES`
The output argument type is `int*`.

`SCHA_DESIRED_PRIMARIES`
The output argument type is `int*`.

`SCHA_FAILBACK`
The output argument type is `boolean_t*`.

`SCHA_RESOURCE_LIST`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_STATE`
The output argument type is `scha_rgstate_t*`.

`SCHA_RG_DEPENDENCIES`
The output argument type is `scha_str_array_t**`.

`SCHA_GLOBAL_RESOURCES_USED`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_MODE`
The output argument type is `rgm_rgmode_t*`.

`SCHA_IMPLICIT_NETWORK_DEPENDENCIES`
The output argument type is `boolean_t*`.

`PINGPONG_INTERVAL`
The output argument type is `int*`.

`SCHA_PATHPREFIX`
The output argument type is `char**`.

`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. It returns the state of the resource group on that node.

`SCHA_RG_PROJECT_NAME`
The output argument type is `char**`.

RETURN VALUES

These functions return 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_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\n", 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 on 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_RG_DESCRIPTION`
The output argument type is `char**`.

`SCHA_NODELIST`
The output argument type is `scha_str_array_t**`.

`SCHA_MAXIMUM_PRIMARIES`
The output argument type is `int*`.

`SCHA_DESIRED_PRIMARIES`
The output argument type is `int*`.

`SCHA_FAILBACK`
The output argument type is `boolean_t*`.

`SCHA_RESOURCE_LIST`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_STATE`
The output argument type is `scha_rgstate_t*`.

`SCHA_RG_DEPENDENCIES`
The output argument type is `scha_str_array_t**`.

`SCHA_GLOBAL_RESOURCES_USED`
The output argument type is `scha_str_array_t**`.

`SCHA_RG_MODE`
The output argument type is `rgm_rgmode_t*`.

`SCHA_IMPLICIT_NETWORK_DEPENDENCIES`
The output argument type is `boolean_t*`.

`PINGPONG_INTERVAL`
The output argument type is `int*`.

`SCHA_PATHPREFIX`
The output argument type is `char**`.

`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. It returns the state of the resource group on that node.

`SCHA_RG_PROJECT_NAME`
The output argument type is `char**`.

RETURN VALUES

These functions return 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_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\n", 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_err_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 on 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 that the resource is configured in. 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 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 information. No value is returned for the out parameter 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 a subsequent calls.

`scha_resource_close()` takes a *handle* argument 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

The macros naming 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)).

SCHA_DESCRIPTION
The output argument type is char**.

SCHA_TYPE
The output argument type is char**.

SCHA_RESOURCE_PROJECT_NAME
The output argument type is char**.

SCHA_ON_OFF_SWITCH
The output argument type is scha_switch_t*.

SCHA_MONITORED_SWITCH
The output argument type is scha_switch_t*.

SCHA_RESOURCE_STATE
The output argument type is scha_rsstate_t*.

SCHA_CHEAP_PROBE_INTERVAL
The output argument type is int*.

SCHA_THOROUGH_PROBE_INTERVAL
The output argument type is int*.

SCHA_RETRY_COUNT
The output argument type is int*.

SCHA_RETRY_INTERVAL
The output argument type is int*.

SCHA_FAILOVER_MODE
The output argument type is scha_failover_mode_t*.

SCHA_RESOURCE_DEPENDENCIES
The output argument type is scha_str_array_t**.

SCHA_RESOURCE_DEPENDENCIES_WEAK
The output argument type is scha_str_array_t**.

SCHA_NETWORK_RESOURCES_USED
The output argument type is scha_str_array_t**.

SCHA_SCALABLE
The output argument type is boolean_t*.

SCHA_PORT_LIST
The output argument type is scha_str_array_t**.

SCHA_LOAD_BALANCING_POLICY
The output argument type is char**.

`SCHA_LOAD_BALANCING_WEIGHTS`
: The output argument type is `scha_str_array_t**`.

`SCHA_AFFINITY_TIMEOUT`
: The output argument type is `int*`.

`SCHA_WEAK_AFFINITY`
: The output argument type is `boolean_t*`.

`SCHA_UDP_AFFINITY`
: The output argument type is `boolean_t*`.

`SCHA_STATUS`
: The output argument type is `scha_status_value_t**`.

`SCHA_START_TIMEOUT`
: The output argument type is `int*`.

`SCHA_STOP_TIMEOUT`
: The output argument type is `int*`.

`SCHA_VALIDATE_TIMEOUT`
: The output argument type is `int*`.

`SCHA_UPDATE_TIMEOUT`
: The output argument type is `int*`.

`SCHA_INIT_TIMEOUT`
: The output argument type is `int*`.

`SCHA_FINI_TIMEOUT`
: The output argument type is `int*`.

`SCHA_BOOT_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_MONITOR_CHECK_TIMEOUT`
: The output argument type is `int*`.

`SCHA_PRENET_START_TIMEOUT`
: The output argument type is `int*`.

`SCHA_POSTNET_STOP_TIMEOUT`
: The output argument type is `int*`.

`SCHA_NUM_RG_RESTARTS`
: The output argument type is `int*`.

`SCHA_NUM_RESOURCE_RESTARTS`
: The output argument type is `int*`.

SCHA_TYPE_VERSION
: The output argument type is char**.

SCHA_STATUS_NODE
: The output argument type is scha_status_value_t**. An additional argument type is char*. The additional argument names the node. It returns the status of the resource on that node.

SCHA_RESOURCE_STATE_NODE
: The output argument type is scha_rsstate_value_t*. An additional argument type is char *. The additional argument names the node. It returns the state of the resource on that node.

SCHA_EXTENSION
: The output argument type is scha_extprop_value_t**. An additional argument type is char*. The additional argument names an extension property of the resource. It returns a structure containing the type and value of the property.

SCHA_ALL_EXTENSIONS
: The output argument type is scha_str_array_t**. It returns the names of all extension properties of the resource.

SCHA_GROUP
: The output argument type is char**. It returns the name of the resource group that the resource is configured in.

The macros naming 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_RT_DESCRIPTION	The output argument type is char**.
SCHA_RT_BASEDIR	The output argument type is char**.
SCHA_SINGLE_INSTANCE	The output argument type is boolean_t*.
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_FAILOVER	The output argument type is boolean_t*.
SCHA_API_VERSION	The output argument type is int*.
SCHA_RT_VERSION	The output argument type is char**.
SCHA_PKGLIST	The output argument type is scha_str_array_t**.
SCHA_START	The output argument type is char**.
SCHA_STOP	The output argument type is char**.
SCHA_VALIDATE	The output argument type is char**.
SCHA_UPDATE	The output argument type is char**.

`SCHA_INIT`	The output argument type is `char**`.
`SCHA_FINI`	The output argument type is `char**`.
`SCHA_BOOT`	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_MONITOR_CHECK`	The output argument type is `char**`.
`SCHA_PRENET_START`	The output argument type is `char**`.
`SCHA_POSTNET_STOP`	The output argument type is `char**`.
`SCHA_IS_LOGICAL_HOSTNAME`	The output argument type is `boolean_t*`.
`SCHA_IS_SHARED_ADDRESS`	The output argument type is `boolean_t*`.

RETURN VALUES

These functions 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_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;
    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);

    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,
```

EXAMPLE 1 Using the `scha_resource_get()` Function *(Continued)*

```
          retry_count_out);

    printf("The log level for resource %s is %d\n", resource_name,
          loglevel_out->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 naming resource type properties are listed below. The value of the named property of the resource's type is output.

`SCHA_RT_DESCRIPTION`
The output argument is of type `char **`.

`SCHA_RT_BASEDIR`
The output argument is of type `char **`.

`SCHA_SINGLE_INSTANCE`
The output argument is of type `boolean_t **`

`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_FAILOVER`
The output argument is of type `boolean_t **`

`SCHA_API_VERSION`
The output argument is of type `int*`.

`SCHA_RT_VERSION`
The output argument is of type `char **`.

`SCHA_PKGLIST`
The output argument is of type `scha_str_array_t **`.

`SCHA_START`
The output argument is of type `char **`.

`SCHA_STOP`
The output argument is of type `char **`.

`SCHA_VALIDATE`
The output argument is of type `char **`.

`SCHA_UPDATE`
The output argument is of type `char **`.

`SCHA_INIT`
The output argument is of type `char **`.

`SCHA_FINI`
The output argument is of type `char **`.

`SCHA_BOOT`
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_MONITOR_CHECK`
The output argument is of type `char **`.

`SCHA_PRENET_START`
The output argument is of type `char **`.

`SCHA_POSTNET_STOP`
The output argument is of type `char **`.

`SCHA_IS_LOGICAL_HOSTNAME`
The output argument is of type `boolean_t **`

`SCHA_IS_SHARED_ADDRESS`
The output argument is of type `boolean_t **`.

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 naming resource type properties are listed below. The value of the named property of the resource's type is output.

`SCHA_RT_DESCRIPTION`
The output argument is of type `char **`.

`SCHA_RT_BASEDIR`
The output argument is of type `char **`.

`SCHA_SINGLE_INSTANCE`
The output argument is of type `boolean_t **`

`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_FAILOVER`
The output argument is of type `boolean_t **`

`SCHA_API_VERSION`
The output argument is of type `int*`.

`SCHA_RT_VERSION`
The output argument is of type `char **`.

`SCHA_PKGLIST`
The output argument is of type `scha_str_array_t **`.

`SCHA_START`
The output argument is of type `char **`.

`SCHA_STOP`
The output argument is of type `char **`.

`SCHA_VALIDATE`
The output argument is of type `char **`.

`SCHA_UPDATE`
The output argument is of type `char **`.

`SCHA_INIT`
The output argument is of type `char **`.

`SCHA_FINI`
The output argument is of type `char **`.

`SCHA_BOOT`
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_MONITOR_CHECK`
The output argument is of type `char **`.

`SCHA_PRENET_START`
The output argument is of type `char **`.

`SCHA_POSTNET_STOP`
The output argument is of type `char **`.

`SCHA_IS_LOGICAL_HOSTNAME`
The output argument is of type `boolean_t **`

`SCHA_IS_SHARED_ADDRESS`
The output argument is of type `boolean_t **`.

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 naming resource type properties are listed below. The value of the named property of the resource's type is output.

`SCHA_RT_DESCRIPTION`
: The output argument is of type `char **`.

`SCHA_RT_BASEDIR`
: The output argument is of type `char **`.

`SCHA_SINGLE_INSTANCE`
: The output argument is of type `boolean_t **`

`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_FAILOVER`
: The output argument is of type `boolean_t **`

`SCHA_API_VERSION`
: The output argument is of type `int*`.

`SCHA_RT_VERSION`
: The output argument is of type `char **`.

`SCHA_PKGLIST`
: The output argument is of type `scha_str_array_t **`.

`SCHA_START`
: The output argument is of type `char **`.

`SCHA_STOP`
: The output argument is of type `char **`.

`SCHA_VALIDATE`
: The output argument is of type `char **`.

`SCHA_UPDATE`
: The output argument is of type `char **`.

`SCHA_INIT`
: The output argument is of type `char **`.

`SCHA_FINI`
: The output argument is of type `char **`.

`SCHA_BOOT`
: 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_MONITOR_CHECK`
The output argument is of type `char **`.

`SCHA_PRENET_START`
The output argument is of type `char **`.

`SCHA_POSTNET_STOP`
The output argument is of type `char **`.

`SCHA_IS_LOGICAL_HOSTNAME`
The output argument is of type `boolean_t **`

`SCHA_IS_SHARED_ADDRESS`
The output argument is of type `boolean_t **`.

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 paramtable properties
`;`	Terminates properties and attributes
`=`	Separates property names and property values or attribute names and attribute values
`,`	Separates values in a valuelist

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: ; = , { } 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

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;

#$upgrade
```

EXAMPLE 1 A Sample Registration File *(Continued)*

```
#$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.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 defined by Sun Cluster. `RG_name` is the only required resource group property, see description below.

Resource Group Properties and Descriptions

`Auto_start_on_new_cluster`

This property controls whether the Resource Group Manager starts the resource group automatically when a new cluster is forming. The default is TRUE.

If set to TRUE, the Resource Group Manager 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 will remain offline until the first time that it is manually switched online using `scswitch`(1M) or the equivalent GUI command. After that, it will resume 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`	When disabled

`Maximum_primaries (integer)`
The maximum number of nodes where the group might be online at once.

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

`Nodelist (string array)`
A comma-separated 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 that resources in the group can write essential administrative files in. 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 to come online more than once within the past `Pingpong_interval` seconds on a particular node (because the resource's `START` or `PRENET_START` method exited non-zero or timed out), that node is considered ineligible to host the resource group and the RGM looks for another master.

If a call to a resource's `scha_control` command or function causes the resource group to be brought offline on a particular node within the past `Pingpong_interval` seconds, that node is ineligible to host the resource group as the result of a subsequent call to `scha_control` originating from another node.

`Default`	3,600 (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 when the administrator adds or removes resources from the resource group.

`Default`	The empty list
`Tunable`	Never

`RG_dependencies (string array)`
A comma-separated list of resource groups that this group depends on. This list indicates a preferred order for bringing other groups online or offline on the same node. It has no effect if the groups are brought online on different nodes.

`Default`	The empty list
`Tunable`	Any time

`RG_description (string)`
A brief description of the resource group.

`Default`	The empty list
`Tunable`	Any time

`RG_project_name (string)`
The Solaris project name (see `projects`(1)) 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 and the user `root` must be configured as a member of the named project (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 effect after the resource has been restarted.

`Default`	Name of the system default project
`Tunable`	Any time
`Valid value`	Any valid Solaris project name

`RG_mode (enum)`

Indicates whether the resource group is a failover or 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 have a value greater than 1, meaning the group can be mastered by multiple nodes simultaneously.

`Note:` The RGM does not allow a resource whose `Failover` property is `True` to be added to a resource group whose `RG_mode` is `Scalable`.

`Default:`	`Failover` if `Maximum_primaries` is 1.
`Scalable`	If `Maximum_primaries` is greater than 1.
`Tunable:`	Never

`RG_name (string)`

The name of the resource group. This property is required and must be unique within the cluster.

`Default:`	No Default
`Tunable:`	Never

`RG_state:` on each `cluster` node `(enum)`

Set by the RGM to `Online, Offline, Pending_online, Pending_offline` or `Error_stop_failed` to describe the state of the group on each cluster node. A group can also exist in an unmanaged state when it is not under the control of the RGM.

This property is not user configurable.

`Default:`	Offline
`Tunable:`	Never

SEE ALSO

`projects`(1), `scrgadm`(1M), `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.

Resource Property Values

`Required`	The administrator must specify a value when creating a resource with an administrative utility.
`Optional`	If the 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 system administrators. 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 system administrator using the command:

```
# scrgadm -c -j <resource> -y <property>=<new value>
```

Resource Properties and Descriptions

`Affinity_timeout (integer)`

Length of time in seconds during which connections from a given client IP address for any service in the resource will be sent to the same server node.

`-1`	All connections are sent to the same node.
`0`	All open connections are sent to the same node.
`<n>`	Specified `<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 will be 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 only used 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 administrator 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`.

This property is required if the `Default` attribute is not specified in the property declaration in the RTR file.

`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 on 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)`

Possible settings are None, Soft, and Hard. Controls whether the RGM relocates a resource group or aborts a node in response to a failure of a `START`, `STOP`, or `MONITOR_STOP` method call on the resource. `None` indicates that the RGM should just set the resource state on method failure and wait for operator intervention. `Soft` indicates that failure of a `START` method should cause the RGM to relocate the resource's group to a different node while failure of a `STOP` or `MONITOR_STOP` method should cause the RGM to set the resource to `STOP_FAILED` state and the resource group to `ERROR_STOP_FAILED` state and wait for operator intervention. For `STOP` or `MONITOR_STOP` failures, the `None` and `Soft` settings are equivalent. `Hard` indicates that failure of a `START` method should cause the relocation of the group and failure of a `STOP` or `MONITOR_STOP` method should cause the forcible stop of the resource by aborting the cluster node.

`Category`	Optional
`Default`	None
Tunable	Any time

`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), who connects to an IP address of a wildcard sticky service, is always sent to the same cluster node regardless of the port number it is coming to.

`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	Any time

`method_timeout for each callback method`

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`	Any time

`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 comma-separated list of logical host name or shared address network resources used by the resource. For scalable services, this property must refer to shared address resources that exist in a separate resource group. For failover services, this property refers to logical host name or shared address resources that exist in the same resource group. The RGM automatically creates this property if the `Scalable` property is declared in the RTR file. If `Scalable` 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. Rather, it is set by the RGM to the number of `scha_control RESOURCE_RESTART` and/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. If a resource type does not declare the `Retry_interval` property, then 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. Rather, it is set by the RGM to the number of `scha_control RESTART` calls that have been made by this resource for its containing RG 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, then 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 the protocol being used by that port, for example, Port_list=80/tcp. 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.

For specifics on setting up this property for Apache, see the Apache chapter in the *Sun Cluster 3.1 Data Services Installation*

Category	Conditional/Required
Default	No Default
Tunable	At creation

R_description (string)
A brief description of the resource.

Category	Optional
Default	The empty string
Tunable	Any time

Resource_dependencies (string array)
A comma-separated list of resources in the same group that must be online in order for this resource to be online. This resource cannot be started if the start of any resource in the list fails. When bringing the group offline, this resource is stopped before those in the list. Resources in the list are not allowed to be disabled unless this resource is disabled first.

Category	Optional

`Default`	The empty list
`Tunable`	Any time

`Resource_dependencies_weak (string array)`

A list of resources in the same group that 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 and the `STOP` methods of this resource before the `STOP` methods of those in the list. The resource can still be online if those in the list fail to start. Resources in the list are not allowed to be disabled unless this resource is first disabled.

`Category`	Optional
`Default`	The empty list
`Tunable`	Any time

`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 will use the predefined project name `default`. The specified project name must exist in the projects database and the user `root` must be configured as a member of the named project (see `projects`(1) and *System Administration Guide: Resource Management and Network Services*). This property is only suppored starting in Solaris 9.

Note – Changes to this property take effect after the resource has been restarted.

`Category`	Optional
`Default`	Null
`Tunable`	Any time
`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`, `Stop_failed`, `Start_failed`, `Monitor_failed`, and `Online_not_monitored`.

This property is not user configurable.

`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. This property is created by the RGM and made available to the administrator only if it is declared in the RTR file. It 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`.

This property is required if the `Default` attribute is not specified in the property declaration in the RTR file.

`Category`	Conditional
`Default`	See above
`Tunable`	When disabled

`Retry_interval (integer)`

The number of seconds over 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 administrator only if it is declared in the RTR file. It 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`.

This property is required if the `Default` attribute is not specified in the property declaration in the RTR file.

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. If this property is declared in the RTR file, the RGM automatically creates the following scalable service properties for resources of that type: `Network_resources_used`, `Port_list`, `Load_balancing_policy`, and `Load_balancing_weights`. 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, the cluster administrator 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 desired, because they can be useful in a non-scalable service as well as in a scalable service.

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; and 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 administrator only if it is declared in the RTR file. It 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.

This property is required if the Default attribute is not specified in the property declaration in the RTR file.

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	Anytime

`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 up, for example, due to a fault monitor restart, a resource failover or switchover, or a node rejoining a cluster after failing.
- 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	Anytime

SEE ALSO

`projects`(1), `scrgadm`(1M), `rt_reg`(4), `property_attributes`(5), `rg_properties`(5), `rt_properties`(5)

Sun Cluster 3.1 Data Services Developer's Guide, System Administration Guide: Resource Management and Network Services

NAME

rt_properties – resource type properties

DESCRIPTION

The list below describes the resource type properties 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 or the object that it belongs to 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 it and it 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 it and it 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 it isn't, 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. `Note:` Resource type properties are not updatable by administrative utilities with the exception of `Installed_nodes`, 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.

`API_version (integer)`

The version of the resource management API used by this resource type implementation.

`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
`Default`	No
`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.

`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)`
Indicates the nodes on which the RGM is to call the `INIT`, `FINI`, `BOOT` and `VALIDATE` methods. 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).

`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`	Configurable by cluster administrator
`Default`	All cluster nodes
`Tunable`	Any time

`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 that a resource of this type is dependent on. This method is expected to do `STOP` actions that must be done after the network interfaces are configured down.

`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 that a resource of this type is dependent on. This method is expected to do `START` actions that must be done before network interfaces are configured up.

`Category`	Conditional/Explicit
`Default`	No Default
`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. It must be a complete path, that is, it 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`	The empty string
`Tunable`	Never

`Resource_type (string)`
The name of the resource type.

To view the names of the currently registered resource types, use:

```
scrgadm -p
```

Starting in Sun Cluster 3.1, a resource type name is of the form

```
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 `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 created prior to Sun Cluster 3.1 continue to be of the form:

```
vendor_id.resource_type
```

`Category`	Required
`Default`	The empty string
`Tunable`	Never

`RT_version (string)`
Starting with 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.

`Category`	Conditional/Explicit
`Default`	No default
`Tunable`	Never

`Single_instance (Boolean)`
If `True,` indicates that only one resource of this type can exist in the cluster. Hence, 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 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.

The `Load_balancing_weights` property can be changed at any time, even while the resource is online. `Network_resources_used`, `Port_list`, and `Load_balancing_policy` are set when the resource is created, and cannot be edited afterward. Depending on how the resource type is implemented, these properties might have default values, or you might be required to provide values at resource creation time.

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`, then these network checks are disabled.

Resource Validatation

Whenever a resource having the `Scalable` =TRUE property is created or updated, the RGM validates various resource properties and will reject the attempted update if these are not configured correctly. Among the checks that are performed are the following:

- The `Network_resources_used` property must be non-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 be non-empty. It must contain a list of port/protocol pairs where protocol is either tcp or udp. For example: `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 3.1 Data Services Installation and Configuration Guide

Sun Cluster 3.1 Data Services Developers' Guide

NAME

SUNW.gds – The Generic Data Service (GDS) is a resource type for making simple network-aware applications highly available or scalable.

DESCRIPTION

GDS is a mechanism for making simple network-aware applications highly available or scalable by plugging then into the Sun Cluster resource Group Management (RGM) framework.

The GDS comprises a fully functional Sun Cluster Resource Type complete with callback methods (`rt_callbacks(1HA)`) and a Resource Type Registration file (`rt_reg(4)`).

Standard Properties

Network_resources_used

```
Category: Optional
Default: Null
Tunable: At creation
```

If this property is omitted, the application is assumed to listen on all addresses. This property need not be specified unless the application binds to one or more specific addresses. See `r_properties(5)` for details.

Before creating the GDS resource, a `LogicalHostname` or `SharedAddress` resource must already have been configured in the same resource group as the GDS resource. See *Sun Cluster 3.0 Data Services Installation and Configuration Guide* for information on how to configure a `LogicalHostname` or `SharedAddress` resource.

Port_list

```
Category: Required
Default: No Default
Tunable: At creation
```

List of port numbers that the application listens on. See `r_properties(5)` for details.

Extension Properties

Start_command

```
Category: Required
Default: No Default
Tunable: At creation
Type: String
```

The `start` command starts the application. It must be a complete command line that can be passed directly to a shell to start the application.

Stop_command

```
Category: Optional
Default: Null
Tunable: At creation
Type: String
```

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 this property is omitted, the Generic Data Service stops the application using signals.

Probe_command

```
Category: Optional
Default: Null
Tunable: At creation
Type: String
```

The probe command periodically checks the health of the 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 OK.

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`, can be an integer between 0 (for success) and 100 (for complete failure). The `probe status` can also be a special value of 201 which results in immediate failover of the application unless `Failover_enables` is set to `False`. The `probe status` is used within the GDS probing algorithm to make the decision of restarting the application locally, as opposed to failing the application over 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, it disconnects immediately. If both connect and disconnect succeed, the application is deemed to be running healthily.

Probe_timeout

```
Category: Optional
Default: 30
Tunable: Any time
Type: Integer
```

This property specifies the timeout value in seconds for the probe command.

Child_mon_level

```
Category: Optional
Default: -1
Tunable: At creation
Type: Integer
```

This property provides control over which processes get monitored through PMF. It denotes the level up to which the forked children processes are monitored. This is similar to the `-C` argument to the `pmfadm` command. Omitting this property, or setting it to the default value of `-1`, has the same effect as omitting the `-C` option on the `pmfadm` command; that is, all children (and their descendents) will be monitored. See the `pmfadm(1M)` man page for more details.

Failover_enabled

```
Category: Optional
Default: true
```

```
Tunable: When disabled
Type: Boolean
```

This property allows the failover of the resource. If the property is set to `false`, failover of the resource is disabled. This can be used to prevent the application resource from initiating a failover of the resource group.

Stop_signal

```
Category: Optional
Default: 15
Tunable: When disabled
Type: Integer
```

This property specifies the signal used to stop the application. The values of this property are the same as those defined in `signal(3head)`.

EXAMPLES

The following sequence of commands illustrates how to make a given application, named `app`, highly available using the GDS. The SunPlex Agent Builder (`scdsbuilder(1HA)`) GUI tool can be used to create driving scripts that contain these commands.

Basic Example

```
# Register the SUNW.gds resource type
scrgadm -a -t SUNW.gds

# Create a resource group for the application
scrgadm -a -g rg1

# Create the LogicalHostname resource
# for the logical hostname 'hhead'
scrgadm -a -L -g rg1 -l hhead

# Create the application resource
scrgadm -a -t SUNW.gds -g rg1 -j app-rs \
        -x Start_command="/usr/local/app/bin/start" \
        -y Port_list="1234/tcp"

# Manage the resource group, enable all the resources, and bring
# them online
scswitch -Z -g rg1

# At this point, the application will be up and running in a highly
# available fashion being monitored by the simple probe provided by
# GDS. The status of the application can now be checked using the
# following command:
scstat -g rg1
```

Complex Example

```
# Register the SUNW.gds resource type
scrgadm -a -t SUNW.gds

# Create a resource group for the application
scrgadm -a -g rg1

# Create the LogicalHostname resource
# for the logical hostname 'hhead'
scrgadm -a -L -g rg1 -l hhead
```

```
# Create the application resource
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

# Manage the resource group, enable all the resources, and bring
# them online
scswitch -Z -g rg1

# At this point, the application will be up and running in a highly
# available fashion being monitored by the fault monitor specified
# on the Probe_command. The status of the application can now be
# checked using the following command:
scstat -g rg1
```

ATTRIBUTES

See `attributes(5)` for a description 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)`

Sun Cluster 3.0 Data Services Installation and Configuration Guide

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

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 `/usr/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.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

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)

NAME

sctransp_rsm – configure the RSM cluster transport

DESCRIPTION

`rsm` is a supported cluster transport type.

SEE ALSO

`scconf`(1M), `scinstall`(1M)