RSF-1 Agent Guide

High-Availability.Com Limited

Suite B,
Pentland House,
Village Way,
Wilmslow,
Cheshire,
SK9 2GH,
United Kingdom.

http://www.high-availability.com

Normal hours	(09:00 – 17:00 GMT)	+44 (0)844 736 1434
Outside hours	(17:00 – 09:00 GMT)	+44 (0)844 736 1974

Issue 1.1, last updated 04/03/2013 15:21:00

Table of Contents

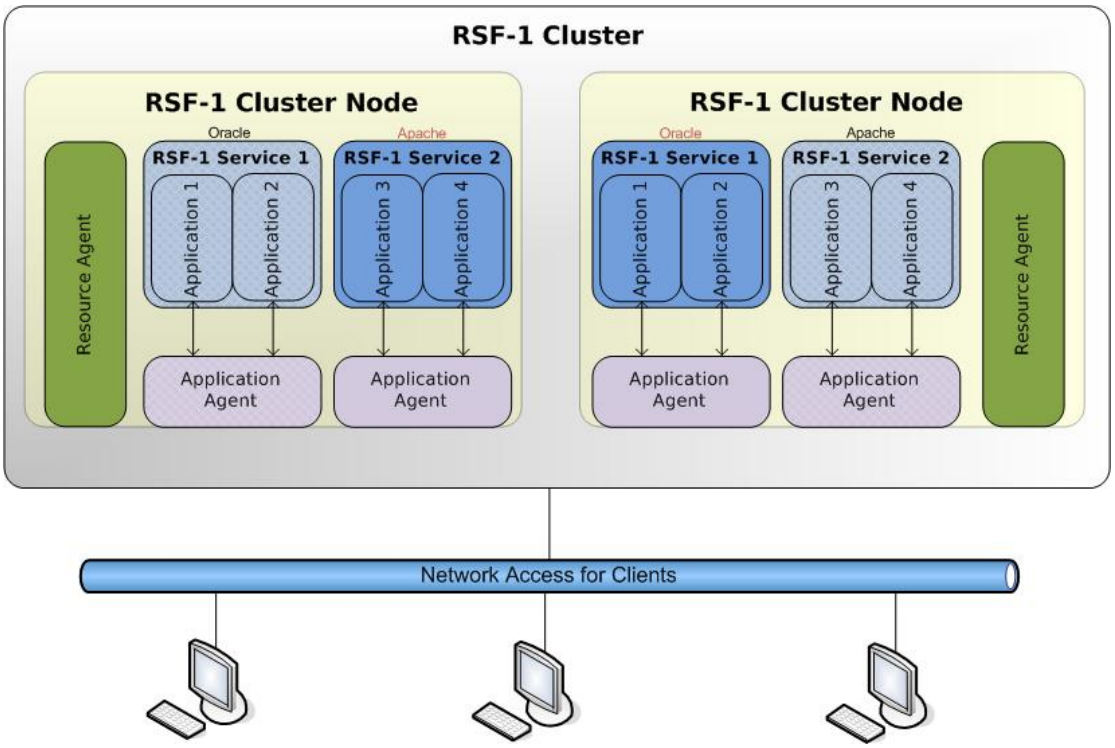
1 Introduction

1.1 Overview

The **rsfagent** is a utility suite for monitoring the health of applications and resources running in an RSF-1 clustered environment. There are two main types of agent, an **application agent** that monitors the health of applications on an RSF-1 cluster node and a **resource agent** that monitors the health of various resources on an RSF-1 cluster node. Collectively the two type of agent form the **RSF-1 Agent Framework**. Figure 1 depicts a conceptual model of this framework.

Operation of the agent framework is not restricted to a single application or resource; it is able to monitor the status of numerous applications and resources simultaneously. Furthermore, the action taken on detection of a failure can be varied on an individual basis

Figure 1 Application agent conceptual model



1.1.1 The application agent

The application agent is aimed at monitoring the health of applications running as part of an RSF-1 **service**. Monitoring is achieved by periodically querying applications to test their availability and functionality. The agent supports the querying of applications in a number of different ways, such as via a TCP port, an SQL database connection for database reads and writes, running an external command and testing its return code; Table 1 on Page 11 lists the available options.

With any test performed, should a response indicate that the application has failed then the agent attempts a recovery procedure. This procedure can vary from restarting the service on the same node, issuing a warning email to an administrator, up to performing a complete service failover to another node in the cluster.

As depicted in Figure 1, a separate copy of the application agent is run for each service being protected; the agent is said to be bound to that service. In this manner, should a service fail over from one node to the other, the agent itself will also be failed over with the service. This ensures that the fail over of one service does not effect the monitoring of any other services on the original node.

1.1.1.1 Differences when monitoring a database

Normally the application agent performs tests on an application by constructing a connection to the application, performing the test, then tearing down the connection – this is known as **connection mode**, and is particularly suited to application availability testing as each test instigated performs a fresh connection every time, and thus tests the complete connection process.

However there are times when once a connection has been established, it is desirable the session should be maintained and application testing is done via this established path, this is known as **session mode**.

Session mode is typically used in conjunction with connection mode when monitoring an application such a database. An initial connection is established and then used for continued monitoring of the database (session mode). However, as a separate process may handle incoming connections (for example the oracle listener), then a complete connection and disconnection should also be performed each testing cycle (connection mode). This provides the flexibility such that should the listener fail then action can be taken on the listener (usually a restart) that will not affect existing connections to the database. If however the database fails in some way (

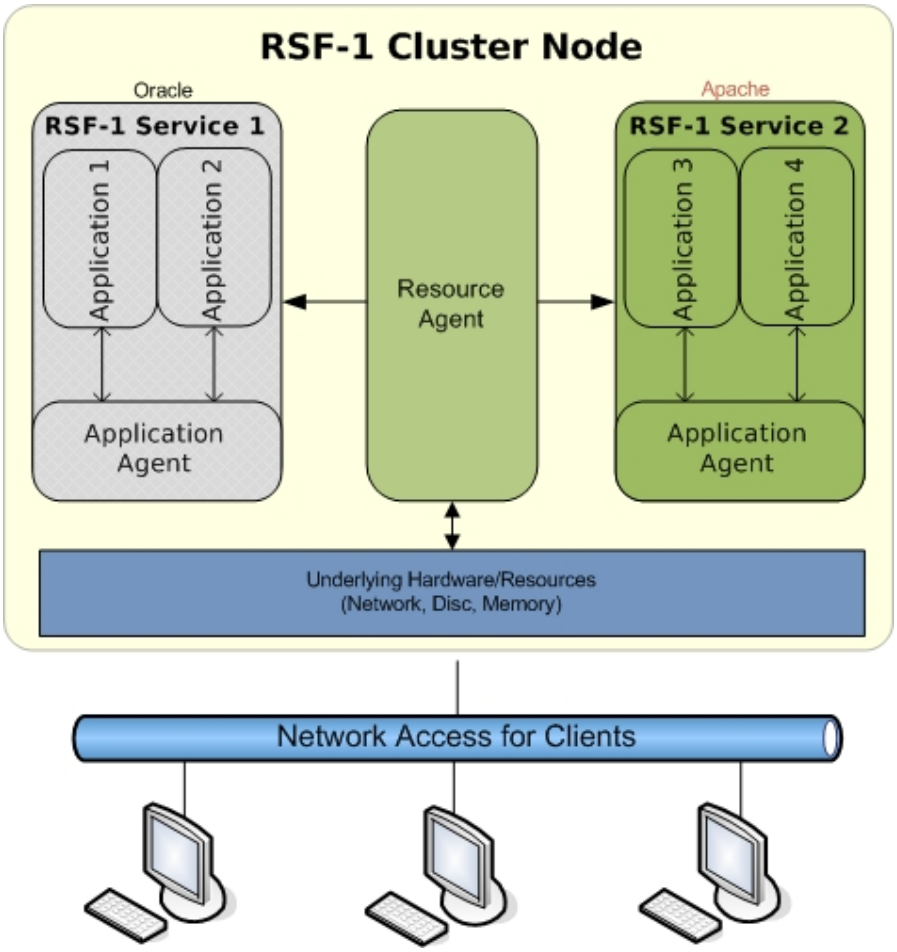
or example it is possible to configure the agent to fail over a database should the session agent detect a failure (which could be the n^{th} failure in a sliding time window), but perform only a local restart should the connection agent detect failure, thus protecting existing logged on users from listener errors.

1.1.2 The resource agent

The resource agent is different from the application agent in that rather than being bound to a particular service, it is bound to a cluster node. In this way it is able to monitor the health of resources on the node and make decisions on the suitability of that node for particular services.

For example, in a two node cluster a resource agent is run on each node to monitor the health of the network connection. The cluster has two services, Oracle running on the first node and Apache running on the second node. Both services make use of the network device eth0[1]. If the resource agent on the second node detects a failure on its network device it will instigate a fail over of the Apache service to the first node. Furthermore, it will also prevent any services that utilise the network device[2] from failing over onto that second node by blocking them; of course should the device recover then any services blocked by the agent are subsequently unblocked[3].

[1] This is an example device name and is specific to Linux; however there are no restrictions to the network device name that can be configured for monitoring, i.e. hme0, bg0, le1 etc.
[2] RSF-1 supplies this information dynamically to the agent.
[3] The amount of time that a resource needs to be a stable, recovered state before services are unblocked is defined in the configuration file.



depicts where the application agent itself fits within the overall RSF-1 cluster. The model depicts a two node cluster with the services Standby (running on node A) and Live (running on node B).

2 Configuration and running the agent

2.1 Log file

The agent log is located in the directory /opt/HAC/RSF-1/agents/log/ in file <*servicename*>.log. The agent logs all messages to this file, including start up, error messages, application failures and application restarts.

When the agent is started it checks for the existence of a previous log file and if found will rename this to <*servicename*>.log.0 before creating a new log file. Log file rotation is performed for up to ten copies of previous logs.

2.2 Starting the agent

The agent should be started as the last part of a RSF-1 service using the script S90rsfagent. This script is located as `/opt/HAC/RSF-1/agents/scripts/S90rsfagent`. For each service to be monitored, copy this script to the service script directory. For example a service called afs would have the service script directory `/opt/HAC/RSF-1/etc/rc.afs.d/` into which the `S90rsfagent` script should be copied. Once the script is installed run the command `/opt/HAC/RSF-1/bin/rsfklink` to create appropriate kill (K) scripts from the start scripts.

The S90rsfagent script ultimately calls /opt/HAC/RSF-1/bin/rsfagent with an argument to provide a postfix for a particular service. This postfix is then used by the agent to:

- Build a service specific log file name.
- Uniquely identify the process ID of the agent running for this service.
- Build the file name of the configuration file specific for this service.

Using a postfix for log and configuration files allows multiple copies of the agent to be run on the same host, one for each RSF-1 service with applications to be monitored.

2.3 Configuration

All of the monitoring details for the agent as a whole are held in a single configuration file /opt/HAC/RSF-1/agents/etc/config.<*servicename*>. The configuration tells the agent what to monitor and what actions to take upon failure of the monitoring. Any number of services to be monitored can be specified in this single file along with options specific to the agent itself. It also contains details of what RSF-1 service each application being monitored belongs to (allowing application failure to trigger RSF-1 service failover if required).

The configuration file contains single line keyword entries:

group.key=value

The fields of the line are:

group

Groups together a number of key value pairs to apply to a single monitored service. The group keyword **option** is a reserved word and is used to introduce options specific to the agent itself.

keyword

A configuration keyword.

value

Assigns a value to the keyword, allowable types are:

- **STRING**

  The alphabetic characters "**a-z _-.!/@ A-Z"** For values with spaces, enclose them in quotation marks ("").

- **PASSWORD**

  An encoded string for use in password fields (see section 2.5 Creating passwords for the configuration file for details on how to produce this string).

- **INT**

  An integer represented by the numerals **0-9**.

- **SECONDS**

  A value in seconds represented by the numerals **0-9**.

- **BOOL**

  A specific string value to indicate a feature is enabled or disabled. Allowable values are **true** or **false**.

A hash (#) symbol at the beginning of a line indicates a comment; the complete line will be ignored. Any errors encountered whilst reading the configuration file are written to the log. If the error is a fatal one (for example a missing mandatory field) then the agent will terminate. Once the configuration file is read in, any groups that are missing mandatory keywords will be discarded and a message detailing the missing entries made in the log file.

The following sections document the available keywords. The complete specification syntax of a keyword declaration is:

<*group*>.**keyword**=<*type*>

Where <*group*> is replaced with the group name and <*type*> is replaced with a value having the same type as that specified in the declaration (the allowable types are listed above). For example:

```
httpd.agenttype=tcp
httpd.description=”Apache service for library intranet”
httpd.host=localhost
httpd.port=80
```

2.3.1 Mandatory fields

A number of fields for each group are mandatory and must be supplied (for example the type of monitoring to be performed). These entries are listed below along with a description of their function. Should any of these fields be missing in a configuration for any group then the agent logs the error that occurred and terminates with no further processing.

<GROUP>.**AGENTTYPE**=*<STRING>*

Sets the type of monitoring that will be performed for this group. The supported types of monitoring available are shown in Table 1 below.

Table 1 Available agent monitoring types

Value	Monitoring type
TCP	Monitors a specific TCP port that an application provides service on. The agent connects to this port on a regular basis to ensure the application is still responding. UDP monitoring is unsupported due to its very nature (i.e. no acknowledgement of packets received and thus no way to detect packet failure vs. legal packet dropping by the application).
CMD	Monitors applications using specific command(s). Should be used when an application provides its own means to interrogate the health of the service it provides.
DBSESSION	A database session agent - connects to and then monitors the database health. In session mode the agent maintains a connection to a database

	and periodically writes and reads back data.
DBCONNECT	Provides the same functionality as the database session agent, however in connect mode the agent repeatedly reconnects to the database each time data is written and then read back. This mode of operation continually tests the ability to create a new connection to the database as opposed to the session agent that creates and utilises a single connection.

<GROUP>.**HOST**=*<STRING>*

The host name of the machine to connect to which is running this service. Usually this is set to `localhost`.

<GROUP>.**PORT**=*<INT>*

The port number the application listens on. The agent connects to this port on the host specified by the `host` keyword.

<GROUP>.**FREQUENCY**=*<SECONDS>*

The frequency by which the application should be monitored. This value represents the time between each test cycle.

<GROUP>.**STARTSCRIPT**=*<STRING>*

Fully qualified path to the location of an executable program or script used to start the application.

<GROUP>.**STOPSCRIPT**=*<STRING>*

Fully qualified path to the location of an executable program or script used to stop the application.

<GROUP>.**WAITAFTERSTART**=*<SECONDS>*

The number of seconds after starting a service to wait before monitoring restarts. Some applications take time before they are fully started and can service requests (and therefore can be successfully monitored).

2.3.2 Optional configuration fields

Although groups have a bare minimum of required fields (as described above), a number of optional fields exist that modify or enhance the way an application is monitored.

<GROUP>.**DESCRIPTION**=*<STRING>*

Assigns a meaningful description to this group. It is used in log file entries and the RSF-1 GUI.

<GROUP>.**MAXRESTARTS**=*<INT>*

The maximum number of restarts in a period of time (see `RESTARTPERIOD`) before the service will be deemed to have failed on this node and a failover to another node is instigated. When no maximum is specified then the agent continually restarts the application whenever a failure is detected

<GROUP>.**RESTARTPERIOD**=*<INT>*

Defines a period of time implemented as a sliding window within which to apply to the maximum number of application restarts given in the `MAXRESTARTS` field. When no period is given then the restart period is taken to be infinite (and in effect maxrestarts is ignored).

<GROUP>.**SYNCSTARTTIMEOUT**=*<INT>*

Specifies at least one successfully monitor event of the application being tested should be made before monitoring failures are acted upon. This allows applications with slow or

variable start up sequences (for example a database performing consistency checking) not to cause a failure due to initial non response to the monitor. The timeout specified is the maximum amount of time to wait for the application to respond before a warning message is emailed to the administrator.

Once the timeout expires the monitor, by default, continues waiting for the application to respond – this action can be changed using the SYNCTARTTIMEOUTABORT setting.

<GROUP>.**SYNCSTARTTIMEOUTABORT**=*<BOOLEAN>*

If set to true once the timeout for SYNCSTARTTIMEOUT is reached the monitor is disabled rather than continuing to wait for the application to become enabled.

<GROUP>.**SERVICE**=*<STRING>*

Specifies the RSF-1 service this application belongs to. This is needed when service failover is desired (both MAXRESTARTS and RESTARTPERIOD are set) so the agent can issue the appropriate failover commands (i.e. to the appropriate RSF-1 service).

<GROUP>.**RUNSTOPBEFORESTART**=*<BOOLEAN>*

A value other than -1 will cause the stop script to be run before the start script is run when a failure is detected (and the application is to be restarted on this node). Running the stop script before the start script may be desirable where a clean shutdown needs to be guaranteed. The default value is -1 (i.e. no stop script).

<GROUP>.**ATTEMPTSBEFOREFAILURE**=*<INT>*

Number of failed TCP connections attempts made to the service before it is deemed to have failed. If not specified then the default is set to 1 attempt.

<GROUP>.**TERMINATESTOPSCRIPT**=*<SECONDS>*

If the STOPSCRIPT is being run and it is active for more than the seconds specified in this field then it will be forcibly terminated before running the STARTSCRIPT (unless ABORTIFSTOPFAILS is set to true, in which case monitoring will be terminated).

<GROUP>.**ABORTIFSTOPFAILS**=*<BOOL>*

If this option is set to true, then should the stop script fail to complete successfully the monitor will not attempt to start the application (using the start script). Instead it halts any further monitoring and logs an error in the log file.

<GROUP>.**WAITAFTERSTART**=*<INT>*

The number of seconds to wait after the STARTSCRIPT has successfully started an application before monitoring commences. Allows time for an application to initialise itself, bind to ports, start services etc.

<GROUP>.**MAILONFAIL**=*<BOOL>*

When set, any errors encountered during monitoring of this specific application cause an electronic mail message to be sent to the recipient configured to receive notification messages. This option obviously depends upon the correct configuration for email delivery (see section 2.3.7 - Configuring for email delivery).

<GROUP>.**DISABLE**=*<BOOL>*

When set to true disables monitoring for this group. This option is specific to the group it is bound to – monitoring continues for other applications defined in the configuration file.

<GROUP>.**INITIALWAIT**=*<SECONDS>*

Specifies an initial time in seconds to wait before monitoring is started for this group. This action maybe desirable when applications need a certain initialisation time before they start responding to requests.

2.3.3 Optional fields to enhance port monitoring

<GROUP>.**PORTSEND**=*<STRING>*

A string to be sent to the application bound to the specified TCP port once a successful network connection has been made. This field is used in conjunction with the portexpect field and is written to the application before the response is read back.

<GROUP>.**PORTEXPECT**=*<STRING>*

A regular expression string to be used for comparison with data read from the port/application once a successful network connection has been made. If the PORTSEND option has been specified then that string will be sent before reading data back from the application and comparing it with the regular expression contained in PORTEXPECT. Note that the pattern matching algorithm used searches for an exact match to the regular expression and will span multiple lines of input. To search for an embedded pattern that may occur anywhere in the input string, use the any character wildcard (`.*`) on either side of the expression.

As an example of the use of these fields consider the following send and expect strings for monitoring a HTTP server:

```
httpd.portsend="HEAD / HTTP/1.0\n\n"
httpd.portexpect=".*HTTP/[0-9]+.[0-9]+ [23][0-9][0-9].*"
```

Note that in the PORTSEND string above, an extra newline has been appended. This is to conform to the HTTP standard which specifies a blank line is used to terminate a request, rather than any specifics of the agent. When constructing the string to send over to an application it is a good idea to manually verify it produces the desired result. An application such as telnet can be used to connect and send the string to the application and view the resulting response (if any).

The string being sent to the http server requests just the heading of the root directory (the `index.*` file) without the actual message-body; the returned header will look something like:

```
HTTP/1.1 200 OK
Date: Tue, 17 Mar 2009 13:18:28 GMT
Server: Apache/2.0.63 (Unix) DAV/2
Last-Modified: Thu, 05 Mar 2009 15:19:14 GMT
ETag: "31da7-5b0-b0e0c080"
Accept-Ranges: bytes
Content-Length: 1456
Connection: close
Content-Type: text/html
```

This regular expression in the PORTEXPECT field is then used to search the returned data for a match. In this case the first line matches and this is therefore considered a successful connection. If however the data returned contained either a 4** or 5** response code (indicative that an error has occurred), then the test is deemed to have failed and counted as an unsuccessful connection.

2.3.4 Fields required using an external command to monitor

The following fields are required when the agent type is set to CMD. Failure to include any of these fields for the group causes the agent to write an error message to the log file and exit.

<GROUP>.**COMMAND**=*<STRING>*

The full path of the command to run in order to test the application status. Any output generated by the command is discarded. Its exit code is compared to that given in the SUCCESSRETURNCODE field to determine success or failure.

<GROUP>.**SUCCESSRETURNCODE**=*<INT>*

The return (exit) status of the command being executed to test the application is compared against this field for success or failure indication. If the two codes agree then the application is deemed to be running successfully, otherwise it is deemed to have failed and the appropriate configured action taken.

2.3.5 Fields required when monitoring a database.

The following fields are required when the agent type is set to either DBCONNECT or DBSESSION. Failure to include any of these fields for the group causes the agent to write an error message to the log file and then exit.

To perform database health monitoring the agent writes and reads data from the database. It is therefore necessary to create a dedicated user in the database for this purpose. Supply this user name and password in the DBNAME and DBPASS fields. The user should have permission to create tables and read and write data to and from them. The actual structures used for monitoring within the tables are created by the agent itself when it first connects; therefore aside from creating a user with the correct access permissions no further work is required from the installer.

The same user can be configured for both the DBCONNECT and DBSESSION agent types simultaneously as different tables are created for each type of monitoring.

<GROUP>.**DBENGINE**=*<STRING>*

The type of database engine being monitored. This is used to identify the specific JDBC driver to load before attempting a connection to the database. All drivers are JDBC type 4 (a native-protocol fully Java technology-enabled driver converts JDBC technology calls into the network protocol used by DBMSs directly). The list of currently supported engine types is:

Engine	Notes
oracle	Uses the oracle JDBC thin driver. Supports Oracle 9,10,11,11i
ingres	Supports Ingres RDBMS, Ingres Enterprise Access and Ingres EDBC databases
mysql	Supports MySQL versions 4.1, 5.0, 5.1 and 6.0 (beta)
postgresql	Postgres database (PostgreSQL) – JDBC driver version 8.3 works with all currently supported PostgreSQL database versions.

<GROUP>.**DBNAME**=*<STRING>*

The name/instance of the database to connect to.

<GROUP>.**DBUSER**=*<STRING>*

The user name to connect to the database specified in DBNAME. By convention this name is `rsfagent` (however any valid user name is supported). This user should be created in the database itself.

<GROUP>.**DBPASS**=*<PASSWORD>*

Provides the password for the database user defined in DBUSER. See section 2.5 “Creating passwords for the configuration file” for details on creating password fields.

2.3.6 Options that act on the agent as a whole.

A number of optional fields are available that effect the overall behaviour of the agent rather than that of any specific group. These fields are introduced with the group name OPTION. This group name is a reserved word and thus only the following field names are recognised, any others are simply silently ignored.

OPTION.DEBUG=*<BOOL>*

Setting this option to true enables debugging for the agent. Detailed real time information on the agents operation is written to standard out.

OPTION.DEBUGTOLOGFILE=*<BOOL>*

When set to true and debugging is enabled, the agent appends debug information to the log file rather than writing it to standard out.

OPTION.RSFHOST=*<STRING>*

In normal operation the agent will attempt to communicate with RSF-1 using the host name `localhost` (i.e. the assumption is made that the agent is running on the same machine as RSF-1 itself).

However, there are certain circumstances where this may not be the desired behaviour, such as when the agent is being run as part of a Solaris secondary zone and RSF-1 itself is running in the master zone (i.e. where Solaris secondary zones are run as RSF-1 services to provide zone failover). This option therefore allows the host name RSF-1 is running on to be set to something other than `localhost`.

OPTION.RSFUSER=*<STRING>*

Before the agent can interact with RSF-1 it needs to connect and authorise to the local RSF-1 server. This process is done during the agent start-up using the default RSF-1 user `_rsfadmin`. However, it is recommended that a new RSF-1 user is created (on all nodes in the cluster) using the utility `rsfpasswd` and configured in the agent using this option.

OPTION.RSFPASSWORD=*<PASSWORD>*

If the RSF-1 default user is changed using the rsfuser option then the password associated with this user is specified using this option. The password is held in the configuration file in encrypted form - see section 2.5 “Creating passwords for the configuration file” for details on creating encrypted password fields.

OPTION.TESTMODE=*<BOOL>*

When set to true this option puts the agent into test mode.

In test mode normal application monitoring is performed (as defined in the configuration file), however, no action is taken when a failure is detected other than:

- If RSF-1 failover is not enabled (or available i.e. no other nodes in the cluster are in automatic mode for the service the applications being monitored belong to) then the first failure detection is logged and no further monitoring is performed. No attempt to restart the application is made.
- If RSF-1 failover is enabled then application failure is continually detected and logged up until the final failure that would trigger a failover. The failover trigger is then logged and no further monitoring is performed on the application. No attempt to perform a failover is made.

2.3.7 Configuring for email delivery

If desired the agent is able to send warning messages and alerts about the applications it is monitoring via email. In order to do this it is necessary to configure email setting for the agent to use – these include the server, a valid user, the type of transport to use etc. using the following options.

OPTION.MAILENABLED=*<BOOL>*

Set to true to enable email integration / reporting for the agent.

OPTION.MAILSERVER=*<STRING>*

Set this option to the name of the mail server to connect to for mail delivery.

OPTION.MAILLOGIN=*<STRING>*

A valid electronic mail user on the mail server authorised for mail delivery.

OPTION.MAILPASSWORD=*<PASSWORD>*

The password for the user specified in the **maillogin** option. The password is held in the configuration file in encrypted form - see section 2.5 "Creating passwords for the configuration file" for details on creating password fields.

OPTION.MAILRCPT=*<STRING>*

This is an email address to which messages generated by the agent should be sent.

OPTION.MAILTRANSPORT=*<STRING>*

This is the mail transport to use when delivering mail. Currently the only supported transport type is SMTP; however, future versions of the agent will provide support for a wider range of mail transports so for forward compatibility this option is necessary.

OPTION.MAILPLAINTEXT=*<BOOL>*

Normally the agent sends MIME messages with embedded HTML. Set this option to true if you only want to receive plain text messages.

OPTION.MAILDEBUG=*<STRING>*

This option turns on debugging for all mail delivery. The output is included in the agent log file.

2.4 Configuring for RSF-1 service failover

Where both MAXRESTARTS and RESTARTPERIOD are specified in the configuration file the agent uses a sliding window of time (the RESTARTPERIOD) within which only a certain number of restarts are tolerated before a failover is instigated. Both MAXRESTARTS and RESTARTPERIOD values are necessary before the agent honours the sliding window policy. If either are missing then the agent simply restarts the application each time a failure is detected. Similarly, if no RSF-1 service is specified then failover is disabled for the applications being monitored.

Service failover will only occur if another node in the cluster is set to automatic mode for the service being monitored (and deemed to have failed). Where no other node will take over a service in the cluster (i.e. is not in automatic mode) then fail over is aborted and application monitoring (and restarting upon failure) is continued.

Should a service failover occur, then monitoring of other applications in that service is suspended before the agent exits (in preparation for service shutdown and fail over to the other node).

2.5 Creating passwords for the configuration file

In order to maintain their integrity, password entries in the configuration file are held as encrypted strings. The encrypted strings are created from plain text passwords using the utility `/opt/HAC/RSF-1/bin/agentpasswd`. The utility prompts for entry of the password, encrypts it, and then prints out the encrypted form suitable for use as a password entry in the configuration file:

```
Enter your password (nothing is echoed):
Re-enter to confirm (nothing is echoed):
Encoded: 35231f759e7bd38374b8d0a5257a7b4032bb5d
```

The string printed after `Encoded:` (starting with the number 3 and ending with the letter d) is the encoding of the password entered. The encoded password is now suitable for use in the configuration file:

```
option.rsfpassword=35231f759e7bd38374b8d0a5257a7b4032bb5d
```

3 Using the agent command line interface (CLI)

Agents can be controlled and monitored from any networked host using the agent command line interface program /opt/HAC/RSF-1/bin/agentcli (the CLI). This section describes firstly how to connect to an agent and then describes the command set available for interrogating and manipulating the agent.

3.1.1 Connecting to an agent

Whenever an agent starts up it creates a request handler thread that listens for incoming connections. The handler is bound to a dynamically allocated port on the machine it is running on. Because of the dynamic nature of this it is not possible for the CLI to know which ports are being used by the agent(s) in advance. However, whenever an agent request handler starts and binds to a port, it also registers this information to RSF-1. Thus in turn the CLI can interrogate RSF-1 instances to ascertain what agents are running, and where.

Specifying to the CLI which agent to connect to has various degrees of granularity; on the one hand the CLI can scan the complete network and offer a choice of agents to connect to, whilst at the other end of the scale, the CLI can be instructed which specific host and agent to connect to.

Running just the CLI with no arguments puts it in discovery mode; here it scans all available networks for RSF-1 instances, and for each one found interrogates it for a list of registered agents. Once scanning is complete a list of servers/agents is offered for selection:

```
> agentctl
Scanning 4 networks for RSF-1 agents...
 1) ftp@george - FTP Service
 2) web@zippy  - Apache WWW Service
 3) dns@zippy  - DNS Service
Please select an agent to connect to (q to quit):
```

The scan resulted in three agents being identified, ftp on server george and web/dns on server zippy. This choice can be narrowed down by specifying a particular RSF-1 node to connect to and interrogate using the –n <*node*> option. This avoids the necessity to scan all the available networks for RSF-1 nodes:

```
> agentcli –n zippy
 1) web@zippy - Apache WWW Service
 2) dns@zippy - DNS Service
Please select an agent to connect to (q to quit):
```

To connect directly to an agent, specify the RSF-1 node along with the agent name using the –s <name> command line option:

```
> agentcli -n zippy -s web
Agent name:  web
Description: Apache WWW Service
Version:     HEAD
Build:       Wed 18 Mar 2009 11:00:44 GMT
Connected, type help for a list of commands.
>
```

Finally, to issue a command to the agent with no user interaction at all, specify the node and agent name along with the command to run using the –c <*command*> and –m <*monitor*> options:

```
> agentcli -n zippy -s web –c s –m httpd
httpd: suspended
>
```

Note that as an agent can have more than one monitor thread (for example when a monitored service consists of more than one application) then the –m <*monitor*> option allows an individual monitor to be specified. To specify an action for all monitors of an agent, simply drop the monitor specific –m option. The command will be applied to all monitors of the agent.

3.2 Options when running the CLI

This following command line options are recognised by the CLI.

-c <*command*>

Specifies a command to run after the CLI is connected to an agent. The CLI will run the command and then exit.

-l <*message*>

Append the message to the agents log file. This option can be used in conjunction with the –c option to add a description and time stamp when the command is run.

-m <*monitor*>

For agents with more than a single monitoring thread, this option allows a particular thread to be selected. It should be used in conjunction with the –s option. Multiple monitoring threads are typical where a service comprises of more than one application. For example, a web based service may run a httpd server listening on port 80, and a second application control httpd server listening on a secondary port. Another example could be a shared file system with a samba server on port 139 and a NFS server on port 2049, here both servers provide access to the same file system, and are therefore considered part of the same service, but require two distinct monitoring threads within the agent itself.

-n <*node*>

This option specifies a remote node to connect to and scan for services, rather than having to scan the complete network for available servers. If the specified node has just one agent with a single monitoring thread running it will be connected to automatically.

-p <*portnofile*>

When an agent starts on a node, it stores the port it is bound and listening on into the file /opt/HAC/RSF-1/agents/run/<*servicename*>.port. The servicename portion of the path is replaced with the service name of the agent. This file path can be provided to the CLI using the –p option. The CLI will then read the content of the file for the port number and connect directly to the agent. This method of connection only works if the CLI is being run on the same machine as the agent is running on (i.e. the network name will be localhost).

-r

Turn on regular expression testing mode (for example to test portsend and portexpect strings before committing them to a configuration file). The CLI will prompt for a regular expression, followed by a string to test that expression against followed by the result of the test. For example:

```
> agentctl –r
Enter your regex: .*response\ code\ [23][0-9][0-9].*
Enter input string to search: Server response code 245 – OK

Positive regex match in the string “Server response code 245 – OK”
```

```
Enter your regex: .*response\ code\ [23][0-9][0-9].*
Enter input string to search: Server response code 545 – OK


No match found
```

Use <CTRL>-C to terminate the CLI when in regular expression test mode.

`-s <servicename>`

When used in conjunction with the –n *<node>* option specifies which agent service on the node to connect to. This option is only necessary when there is more than one service running on the node.

3.3 Commands available from the CLI

Once the CLI is connected to an agent a short summary is displayed and then the CLI awaits commands to send to the agent itself:

```
agentctl –n zippy
   1) ftp@zippy – FTP service
   2) web@zippy – Apache WWW Service
Please select an agent to connect to (q to quit): 2
Agent name:  web
Description: Apache WWW Service
Version:     3.0.34
Build:       Tue 24 Mar 2009 16:44:09 GMT
Connected, type help for a list of commands.
>
```

The following commands are supported by the CLI.

Command	Description
d	Toggles the current debugging state, from on to off and vice versa.
D	Display if debugging is on or off.
h	Displays a summary of available commands.
i	Summary of running agent – the same as when the CLI first connects.
l <message>	Append the message to the agent log file.
m	Displays a list of configured monitor threads.
q|Q	Disconnect the CLI from the agent.
r <monitor>	Resume the specified monitor, or all monitors if non specified.
s <monitor>	Suspend the specified monitor, or all monitors if non specified.
S	Status display of agent and all monitor threads.
t	Terminate the agent.
v	Show version details of the agent.

3.4 Controlling agents from shell scripts and cron

It may be desirable to control the agent from either shell scripts or cron table entries; for example during an overnight backup responsiveness may be slow for a file system or database and as such it becomes preferable to suspend any monitoring during this activity.

The CLI provides such a mechanism by accepting all the necessary parameters for connecting and issuing a command to an agent and then exiting. To do this, specify the node the agent is running

on with the –n option, the service name with the –s option, and the command to run with the –c option (either the suspend or resume command). The CLI will then connect to the agent, issue the command and then exit. If the agent is running more than one monitor thread, then the command will apply to all threads. To run the command against just one particular monitor thread, specify its name using the –m option.

If for example an agent is running on the node zippy, with a service name of web, the command to suspend all monitoring activity would be:

```
agentctl –n zippy –s web –c s
```

If the web service was in fact running two monitor threads, `httpd1` and `httpd2`, and it was desirable to suspend monitoring only on the `httpd2` thread then the command to run would be:

```
agentctl –n zippy –s web –m httpd2 –c s
```

Similarly, to resume monitoring, the same commands would be issued with the `–c s` option replaced by `–c r` option:

```
agentctl –n zippy –s web –c s
```

```
agentctl –n zippy –s web –m httpd2 –c s
```

These agent control commands can either then be issued from cron or even embedded in shell scripts as a wrapper around the backup command itself:

```
#!/bin/sh
agentctl –n zippy –s web –c s –l “suspending monitoring for backup”
ufsdump 0f /dev/rmt/0
agentctl –n zippy –s web –c r –l “resuming monitoring after backup”
```

4 Example configurations

A complete set of example configurations are supplied with the agent package in the directory `/opt/HAC/RSF-1/agents/etc/example_configs/`. This section provides a brief overview of two of the most commonly used configurations, that for a HTTP server and a Database server.

4.1.1 Example configuration: HTTP

The following configuration provides monitoring for a HTTP daemon. The first block with the prefix option sets options specific to the agent (and any threads it runs) as a whole.

```
option.mailserver=mail.high-availability.com
option.maillogin=agentconduit
option.mailpassword="3cebe23d1d64ac3f117efd7063626f7e6adca0"
option.mailrcpt=alert@high-availability.com
option.mailtransport=smtp
option.maildebug=no
option.mailenabled=yes
option.mailonfail=yes
option.mailplaintext=no
option.nodecrypt=no
option.description="Apache WWW Service"
option.rsf1service=web
option.debug=no
option.debugtologfile=yes
option.rsfuser="_rsfadmin"
option.rsfpassword="01d9f6bc95be3ac9ee0fef98cb3c5bc0324649"

httpd.disabled="no"
httpd.agenttype=tcp
httpd.portsend="HEAD / HTTP/1.0\n\n"
httpd.portexpect=".*HTTP/[0-9]+.[0-9]+ [23][0-9][0-9].*"
httpd.host=zippy
httpd.port=80
httpd.frequency=10
httpd.startscript="svcadm enable svc:/network/http:apache2"
httpd.stopscript="svcadm disable svc:/network/http:apache2"
httpd.terminatestopscript=15
httpd.runstopbeforestart=no
httpd.abortifstopfail=yes
httpd.maxrestarts=2
httpd.restartperiod=60
httpd.attemptsbeforefailure=2
httpd.waitafterstart=10
```

4.1.2 Example configuration: ORACLE

```
option.rsf1service=oracle

session.agenttype=dbsession
session.dbengine=oracle
session.dbname=demo
session.description="Oracle10g Database Session"
session.dbuser=system
session.dbpass=manager
session.host=localhost
session.port=1521
session.frequency=10
session.startscript="su - oracle -c dbstart"
```

```
session.stopscript="su - oracle -c dbstop"
session.runstopbeforestart=yes
session.waitafterstart=10
session.maxrestarts=3
session.restartperiod=3600
session.attemptsbeforefailure=3

connect.agenttype=dbconnect
connect.dbengine=oracle
connect.dbname=demo
connect.description="Oracle10g Database Connection"
connect.dbuser=system
connect.dbpass=manager
connect.host=localhost
connect.port=1521
connect.frequency=30
connect.startscript="su - oracle -c lsnrctl start"
connect.stopscript="su - oracle -c lsnrctl stop"
connect.runstopbeforestart=no
connect.waitafterstart=30
connect.maxrestarts=3
connect.restartperiod=3600
connect.attemptsbeforefailure=3
```