Application Note: Using Agent Builder to Create New Agents

AIX, HP-UX, Linux, Solaris

5.1

June 2009

symantec.

Application Note: Using Agent Builder to Create New Agents

Symantec Corporation
350 Ellis Street
Mountain View, CA 94043
www.symantec.com

Third-party legal notices

Third-party software may be recommended, distributed, embedded, or bundled with this Symantec product. Such third-party software is licensed separately by its copyright holder.

Technical support

Visit http://www.symantec.com/business/support/assistance_care.jsp for product assistance. Use the Knowledge Base search feature to access resources such as TechNotes, product alerts, software downloads, hardware compatibility lists, and our customer email notification service. If you encounter an error when using a product, include the error number preceding the message when contacting Technical Services. You can also use the error number to search for information in TechNotes or documents on the Web site.

Using Agent Builder to Create New Agents

Introduction

This application note describes the procedure to build an agent using Agent Builder.

This document is meant as a reference for users who want to quickly generate agents with separate agent executables and agent type definitions.

About Agent Builder

The Agent Builder provides a quick and easy way to build a new agent and a new resource type for different applications.

Creating different agents for different applications helps you manage a large number of applications independently. You can define attributes that are specific to an application individually at the resource type level without affecting other applications.

The Agent Builder contains a core agent from which the agentbuilder utility generates code for a new agent type. The core agent is similar in functionality to any other application agent. The agent has well-defined attributes, entry points, and design patterns for clustering enterprise applications.

Key features

The agent created using Agent Builder is fully feature compatible with the existing application agent. In addition, the agent has the following enhanced features:

- Based on ACCLib and portable across Veritas Cluster Server and Veritas Cluster Server One (VCS One) on UNIX platforms.
- Regular expression matching for process patterns.
- Configurable monitoring sequence.
- First Failure Data Capture for faster troubleshooting and Root Cause Analysis (RCA).
- Support for Solaris zones.
- Can be used to cluster any application that has well-defined start and stop programs. Due to its enhanced and flexible monitoring sequence, the agent can be used for applications which do not have a fixed command line but a complex command pattern.

What’s new in this release

This release supports the following changes for Agent Builder:

- New options have been added to the Agent Builder:
  - `-appab` to install and uninstall the Agent Builder.
  - `-platform` to specify the platform/architecture while creating an agent using Agent Builder.

Supported software

The Agent Builder supports the following software:

Veritas Cluster Server	VCS 4.1, 5.0
Veritas Cluster Server One	VCS One 2.0 on AIX, HP-UX, Linux, and Solaris
ACC Library	5.1 and later
Operating Systems	■ AIX 5.1, 5.2, 5.3, 6.1 on pSeries ■ HP-UX 11i version 2, HP-UX 11i version 3 ■ Red Hat Enterprise Linux 3.0, 4.0, 5.0 on Intel ■ SUSE Linux Enterprise Server 9, 10 ■ Solaris 8, 9, 10 on SPARC and x86 Note: The agent supports zones on Solaris in both VCS and VCS One environments.

Prerequisites for creating an agent using Agent Builder

Ensure that the following prerequisites are met before you create the agent.

- The application for which an agent is developed must lend itself to being controlled by the agent and must be able to operate in a clustered environment. The following criteria describe an application that can successfully operate in a clustered environment:
  - The application should have a well-defined start program.
  - The application should have a well-defined stop program.
  - The application should have at least one well-defined monitoring method.
    Example: The application's process pattern should be known or the application should maintain PID files.
- The application that is to be made highly available is successfully installed and configured on the system.
- The Agent Builder tool is installed on the system.

See "Installing Agent Builder in a VCS environment" on page 9.

- The latest version of the ACC library is installed on the system.
  To install or update the ACC library package, locate the library and related documentation on the agent CD and in the compressed agent tar file.
  See "About ACC library" on page 8.

About ACC library

The operations for the agent created using Agent Builder depend on a set of Perl modules known as the ACC library. The ACC Library contains common, reusable functions that perform tasks, such as process identification, logging, and system calls.

The library must be installed on each system in the cluster that will run the agent.

For more information on ACC Library, see the *Veritas™ ACC Library Installation Guide*.

Installing ACC library

Install the ACC library on each system in the cluster that runs an agent that depends on the ACC library.

To install the ACC library

1. Log in as superuser.
2. Navigate to the pkgs directory (the pkgs directory on the CD or the pkgs directory that is created by extracting the VCS agent tar file).

   AIX
   ```
   # cd_mount/aix/application/acc_library/\
   vcs/version_library/pkgs
   ```

   HP-UX
   ```
   # cd_mount/hpux/generic/application/\
   acc_library/vcs/version_library/pkgs
   ```

   Linux
   ```
   # cd_mount/linux/generic/application/\
   acc_library/vcs/version_library/rpms
   ```

   Solaris
   ```
   # cd_mount/solaris/dist_arch/application/\
   acc_library/vcs/version_library/pkgs
   ```
   where dist_arch is sparc or sol_x64

3 Install the package. Enter **Yes** if asked to confirm overwriting of files in the existing package.

AIX
```
# installp -ac -d VRTSacclib.rte.bff\
VRTSacclib.rte
```

HP-UX
```
# swinstall -s 'pwd' VRTSacclib
```

Linux
```
# rpm -i\
VRTSacclib-VersionNumber-GA_GENERIC.noarch.rpm
```

Solaris
```
# pkgadd -d . VRTSacclib
```

4 For HP-UX, install the HP-UX patch PHCO_29042, if it is not already installed.

Installing Agent Builder in a VCS environment

Perform the following steps to install Agent Builder.

To install Agent Builder

1 Log in as superuser.

2 Navigate to the directory containing the installer for the platform running in your environment.
For VCS 5.0 and VCS One 2.0, specify *vcs_version* as 5.0.
For VCS 4.x, specify *vcs_version* as 4.1.

AIX
```
# cd_mount/aix/application/AgentBuilder/\
vcs_version/version_agent/pkgs
```

HP-UX
```
# cd_mount/hpux/generic/application/\
AgentBuilder/vcs_version/version_agent/pkgs
```

Linux
```
# cd_mount/linux/generic/application/\
AgentBuilder/vcs_version/version_agent/rpms
```

Solaris
```
# cd_mount/solaris/sparc/application/\
AgentBuilder/vcs_version/version_agent/pkgs
```

3 Install the package:

AIX
```
#installp -ac -d VRTSappab.rte.bff
VRTSappab.rte
```

HP-UX	`# swinstall -s 'pwd' VRTSappab`
Linux	`# rpm -ihv VRTSappab-AgentVersion.rpm`
Solaris	`# pkgadd -d . VRTSappab`

Installing Agent Builder in a VCS One environment

You must install AgentBuilder on the Policy Master node in VCS One. Using the `-system` option of the agentbuilder utility, you can deploy the newly created agents on other systems.

To install Agent Builder in a VCS One environment

1. Mount the Agent Pack software disc on the Policy Master system on which you want to install Agent Builder.
2. Depending on the platform type, navigate to the directory containing the installer for the VCS One agents:

AIX	`# cd aix/high_availability_agents`
HP-UX	`# cd hpux/hpux<os_version>\ /high_availability_agents`
Linux	`# cd linux/dist_arch/\ high_availability_agents` Where dist is the Linux distribution and arch is the architecture.
Solaris	`# cd solaris/dist_arch/\ high_availability_agents` where dist_arch is sparc or sol_x64

3. Enter the command to start the Agent Builder installation.

```
# ./installagpack -appab
```

Building an agent using Agent Builder

Perform the following steps to build an agent using Agent Builder.

To build an agent using Agent Builder

Run the following commands.

VCS 4.x	`# cd/opt/VRTSvcs/bin/AgentBuilder`
VCS 5.0 and VCS One 2.0	`# cd opt/VRTSagents/ha/bin/AgentBuilder`

```
# ./agentbuilder <agent> -base <base_product> \
-platform <platform> -[ssh|rsh] [-verbose] [-zones]\
[-system System1] [-system System2] [...] [-system SystemN]
```

where,

- base_product option specifies the base product, which can take values vcs4, vcs5, or vcsone.
- platform specifies the supported platforms which can take a value from following:
  { solaris | solaris/sparc | solaris/x86 | linux | linux/x86 | aix | aix/rs6000 | hpux }.
- ssh is the secure shell to deploy the agent to remote systems.
- rsh is the remote shell to deploy the agent to remote systems.
- zones option, if specified, enables Solaris zones support.
- system option specifies the nodes on which you want to deploy the agent.

The Agent Builder builds the agent and type as specified in the arguments and copies it to all the nodes as defined by the argument.

For VCS 4.1, the Agent Builder deploys the new agent in the directory /opt/VRTSvcs/bin on the remote system specified by `-system <system name>`.

For VCS 5.0 and VCS One 2.0, the Agent Builder deploys the new agent in the directory /opt/VRTSagents/ha/bin on the remote system specified by `-system <system name>`.

For examples of building an agent using Agent Builder, see “Sample Configurations” on page 21.

Configuring the agent

After the agent and the agent type are created, you can configure the resources using the newly created agent type.

Before you configure the resources, you must first define the attributes and the agent functions or entry points.

Attribute Definitions

The agent created by Agent Builder has the following attributes:

Table 2-1 Agent attributes

Attribute	Definition
CleanProgram String	Complete path to a user-specified utility that is used to forcibly stop the application. Command-line arguments are supported. This is an optional attribute. Example: `/IBMIHS/bin/myclean.sh` Default Value: No default value
EnvFile String	Complete path to the file that the agent for Agent Builder sources to set the environment variables. This is an optional attribute. Example: `/IBMIHS/bin/setEnv.sh` Default Value: No default value
ListenAddressPort String	A composite string containing the virtual hostname/IPv4 address and listen port delimited by a colon (":"). This is used to do a connect during monitoring, to determine if the application is listening to the port on the specified host. This is an optional attribute. Example: `adminsol.veritas.com:9191` Default Value: No default value
MonitorProcessPatterns Vector String	A list of processes to be monitored and cleaned. Process pattern can be a regular expression or a process name with full command-line arguments displayed by the `ps` command. This is an optional attribute. `Example: {"/IBMIHS/bin/httpd.*-f.*/IBMIHS/conf/httd.conf"}` Default Value: No default value

Table 2-1 Agent attributes

Attribute	Definition
MonitorProgram String	Contains the complete path name and command-line arguments for an externally provided monitor program. The monitor entry point executes this program to perform a user-defined state check. Based on the UNIX user defined in the User attribute, this MonitorProgram runs in this user-defined shell. Monitor entry point executes the MonitorProgram according to the given MonitorProgramFrequency. This program is not supplied with the Agent Builder and is externally developed by the user to satisfy unique requirements. The exit code of the program is interpreted by the monitor entry point as follows: ■ 110 or 0 - Agent Builder is ONLINE. ■ 100 or 1 - Agent Builder is OFFLINE. ■ 99 - Agent Builder state is UNKNOWN. ■ Any other value- The Agent Builder state is UNKNOWN. This is an optional attribute. Example 1: `/ibm/myMonitor.sh` Example 2: `/ibm/myMonitor.sh arg1 arg2` Default Value: No default value

Table 2-1 Agent attributes

Attribute	Definition
MonitorProgramFrequency Integer	Used to enable MonitorProgram. The numeric value of MonitorProgramFrequency specifies how often the MonitorProgram must run. ■ 0- indicates never run the MonitorProgram. ■ 1- indicates run MonitorProgram every monitor interval. ■ 2- indicates run MonitorProgram every second monitor interval, and so on. This is an optional attribute. Example: 1 Default Value: 0 **Note:** ■ Exercise caution while setting MonitorProgramFrequency to large numbers. For example, if the MonitorInterval is set to 60 seconds and the MonitorProgramFrequency is set to 100, then the MonitorProgram is executed every 100 minutes, which may not be as often as intended. For maximum flexibility, no upper limit is defined for MonitorProgramFrequency. ■ If MonitorSequence attribute contains only MonitorProgram, then the value of MonitorProgramFrequency attribute must be set to 1.

Table 2-1 Agent attributes

Attribute	Definition
MonitorSequence String	Used to define the sequence in which monitoring methods will be used for monitoring the resource. MonitorSequence is a combination of one or more of the following monitoring methods separated by a space: ■ MonitorProgram ■ ListenAddressPort ■ MonitorProcessPatterns ■ PidFilesPatterns Each method corresponds to an attribute. This is a mandatory attribute. Example: ■ ListenAddressPort MonitorProgram ■ PidFilesPatterns ListenAddressPort MonitorProgram ■ MonitorProcessPatterns MonitorProgram Default Value: "MonitorProcessPatterns PidFilesPatterns ListenAddressPort MonitorProgram" **Note:** ■ At least one of the attributes specified for MonitorSequence should have a non-null value. ■ If MonitorProgram is the first method defined, ensure that MonitorProgram is available locally on each configured node.

Table 2-1 Agent attributes

Attribute	Definition
PidFilesPatterns Vector String	A list of PID files that contain the process IDs to be monitored and cleaned. These files are application-generated files. Each PID file should contain one PID which will be monitored. Specify the complete path of each PID file in the list. Process pattern can be associated with each PID file, if required. Process pattern can be a regular expression or a process name with full command-line arguments displayed by the `ps` command. This is an optional attribute. Example: `{"/var/logs/httpd.pid","httpd.*-f /IBMIHS/conf/httd.conf"}` Default Value: No default value. **Note:** The process ID can change when the process restarts. If the application takes time to update the PID file, the agent's monitor script may return an incorrect result. If this occurs, increase the ToleranceLimit in the resource definition.
ResLogLevel String	Specifies the logging detail performed by the agent for the resource. Valid values are: ■ ERROR - Only logs error messages. ■ WARN - Logs above plus warning messages. ■ INFO - Logs above plus informational messages. ■ TRACE - Logs above plus trace messages. This is very verbose and should only be used during initial configuration or for troubleshooting and diagnostic operations. Example: INFO Default Value: INFO
StartProgram String	Complete path to the user-supplied external utility that is used to start up the application. Command-line arguments are supported. This is a mandatory attribute. Example: `/IBMIHS/bin/apachectl start` Default Value: No default value.

Table 2-1 Agent attributes

Attribute	Definition
StopProgram String	Complete path to the user-supplied external utility that is used to stop the application. Command-line arguments are supported. This is a mandatory attribute. Example: `/IBMIHS/bin/apachectl stop` Default Value: No default value.
User String	UNIX user name used to run the StartProgram, StopProgram, MonitorProgram, and CleanProgram. If the MonitorProgram attribute is specified, the agent uses this user's credentials to run the defined program. The user name must be synchronized across the systems within the cluster. In other words, the user name must resolve to the same UID and have the same default shell on each system in the cluster. Agent entry points use the `getpwnam` function system call to obtain UNIX user attributes. As a result, the user can be defined locally or can be defined in a common repository (for example, NIS, NIS+, or LDAP). The processes specified in the MonitorProcesses and PidFilesPatterns list must run in the context of the specified user. This is an optional attribute. Example: daemon Default Value: "root"

State Definitions

- ONLINE-Indicates the service being monitored is online.
- OFFLINE-Indicates the service being monitored is offline.
- UNKNOWN-Indicates the service operation is in a pending state, or that the agent could not determine the state of the resource.

Agent Functions (Entry points)

The agent brings services online, takes them offline, and monitors their status.

Online

The agent performs the following tasks in an online operation:

- Performs the following check to ensure that the resource is fully offline.

    - If defined, checks the MonitorProcessPatterns to see if any process defined in the pattern is running.
    - If defined, checks the PidFilesPatterns attribute to see if any PIDs defined in the attribute are running.
- If the resource is deemed to be fully online, then returns immediately.
- If the resource is deemed to be fully offline then executes the start routine by calling the user-defined StartProgram.
- If the resource is deemed to be partially online, then cleans up the partial online state by killing the processes and calling CleanProgram to perform cleanup of the partially online resource.
- After cleaning up a partially online resource proceeds with the online by executing the start routine.

Offline

Executes the stop routine by calling the user-defined StopProgram.

Monitor

The agent performs the following tasks in the monitor operation:

- Executes the different monitoring methods according to the user-defined MonitorSequence. The different monitoring methods are as follows:
    - PidFilesPatterns
    - MonitorProcessPatterns
    - ListenAddressPort
    - MonitorProgram

  The definition of all the monitoring methods is described in the attribute section.
- If any of the above method returns offline, then monitor returns offline.
- If any of the above method returns online, continues with the next method in the sequence defined by the user until all the monitoring methods have been executed. If all of the methods return online, only then is the resource considered to be online.
- The agent runs the MonitorProgram at a periodic interval determined by the MonitorProgramFrequency. The agent runs all other monitoring methods on every monitoring cycle.

Clean

The agent performs the following tasks in the clean operation:

- Calls the user-defined CleanProgram to cleanup any application state, such as stale ipc resources.
- If any of the PIDs or patterns defined in the PidFilesPatterns are found, then clean those.
- If any of the patterns defined in MonitorProcessPatterns are found then clean those.

Resource type definition

The Agent Builder creates new resource type using the resource type definition, which depends on the base product option.

The resource type definition specific to VCS 5.0 is as follows.

```
type AgentBuilder (
    static str AgentFile = "/opt/VRTSvcs/bin/Script50Agent"
    static str AgentDirectory =
    "/opt/VRTSagents/ha/bin/AgentBuilder"
    static str ArgList[] = { ResLogLevel, State, IState,
StartProgram, StopProgram, CleanProgram, MonitorProgramFrequency,
MonitorProgram, User, EnvFile, MonitorSequence, ListenAddressPort,
PidFilesPatterns, MonitorProcessPatterns }
    str ResLogLevel = "INFO"
    str StopProgram
    str MonitorProgram
    str CleanProgram
    str User = "root"
    str EnvFile
    str ListenAddressPort
    str StartProgram
    str MonitorProcessPatterns[]
    int MonitorProgramFrequency = 0
    str PidFilesPatterns{}
    str MonitorSequence = "MonitorProcessPatterns PidFilesPatterns
ListenAddressPort MonitorProgram"
)
```

The resource type definition specific to VCS 4.1 is as follows.

```
type AgentBuilder (
    static str ArgList[] = { ResLogLevel, State, IState,
StartProgram, StopProgram, CleanProgram, MonitorProgramFrequency,
MonitorProgram, User, EnvFile, MonitorSequence, ListenAddressPort,
PidFilesPatterns, MonitorProcessPatterns }
    str ResLogLevel = "INFO"
    str StopProgram
    str MonitorProgram
    str CleanProgram
    str User = "root"
    str EnvFile
    str ListenAddressPort
```

```
    str StartProgram
    str MonitorProcessPatterns[]
    int MonitorProgramFrequency = 0
    str PidFilesPatterns{}
    str MonitorSequence = "MonitorProcessPatterns PidFilesPatterns
ListenAddressPort MonitorProgram"
)
```

Sample Configurations

Following are examples of creating agents using Agent Builder and their sample configurations.

Example 1

In this example, a new agent ABSendmail is created for base product VCS 4.x and it is deployed on system Linux-1.

```
# ./agentbuilder ABSendmail -base vcs4 -platform linux -system
Linux-1
Agent [ABSendmail] created successfully!
Using [ssh] as default to deploy agent on [Linux-1]
root@Linux-1's password:
Successfully deployed agent on system : [Linux-1]
```

In the following sample configuration, you configure the executable `sendmail` as StartProgram, StopProgram, and MonitorProgram, with start, stop, and status specified as command line arguments respectively.

Configure the agent for monitoring using following methods: a process specified by `sendmail` in MonitorProcessPatterns attribute, and PID stored in the PidFilesPatterns attribute and MonitorProgram. As no user is specified, the agent uses the `root` user.

```
ABSendmail sendmail_res (
                StartProgram = "/etc/init.d/sendmail start"
                StopProgram = "/etc/init.d/sendmail stop"
                MonitorProgram = "/etc/init.d/sendmail status"
                MonitorProcessPatterns = { sendmail }
                MonitorProgramFrequency = 1
                PidFilesPatterns = { "/var/run/sendmail.pid" = "" }
                )
```

Example 2

In this example, a new agent ABSamba is created for base product VCS 5.0 and it is deployed on systems HP-1 and HP-2.

```
# ./agentbuilder ABSamba -base vcs5 -platform hpux -system HP-1
-system HP-2
Agent [ABSamba] created successfully!
Using [ssh] as default to deploy agent on [HP-1]
```

```
root@HP-1's password:
Successfully deployed agent on system : [HP-1]
Using [ssh] as default to deploy agent on [HP-2]
root@HP-2's password:
Successfully deployed agent on system : [HP-2]
```

In the following sample configuration, you configure the executable `samba` as StartProgram, StopProgram, and CleanProgram, with start, stop, and force stop specified as command-line arguments respectively.

Configure the agent for monitoring using following methods:

- processes having pattern "smbd" and "nmbd" in MonitorProcessPatterns attribute
- PID stored in “smbd.pid” file and corresponding pattern for the process “smbd” specified in PidFilesPatterns attribute
- an executable, `sambaMonitor`, monitors the application and uses “all” as its command line argument. Monitor program executes according to MonitorProgramFrequency specified, which is 2 monitor cycles in this example.

```
ABSamba samba_app (
                StartProgram = "/usr/sbin/samba start"
                StopProgram = "/usr/sbin/samba stop"
                CleanProgram = "/usr/sbin/samba force stop"
                MonitorProgram = "/usr/local/bin/sambaMonitor all"
                MonitorProcessPatterns = { "smbd", "nmbd" }
                MonitorProgramFrequency = 2
                PidFilesPatterns = { "/var/lock/samba/smbd.pid" =
"smbd" }
                )
```

Example 3

In this example, a new agent ABApache_zones is created for base product VCS 4.x and it is deployed on system sun-1. The new agent supports Solaris zones. As the verbose option is used, detailed messages while creating and deploying the agent are displayed.

```
# ./agentbuilder ABApache_zones -base vcs4 -platform solaris
-verbose -zones -system sun-1
Success: Creating Agent directory structure
Success: Generating Agent module
Success: Setting permissions for Agent module
Success: Generating Agent entry point
Success: Setting permissions for Agent entry point
Success: Generating Agent GUI xml
Success: Setting permissions for Agent GUI xml
Generated online, offline, clean entry points
Opening cluster..
Adding resource type [ABApache_zones]
```

```
Closing cluster..
Agent [ABApache_zones] created successfully!
Deploying agent on all nodes::
Using [ssh] as default to deploy agent on [sun-1]
root@sun-1's password:
Successfully deployed agent on system : [sun-1]
```

In the following sample configuration, you configure the executable `httpd` as StartProgram and StopProgram, with start and stop specified as command line arguments respectively.

Configure the agent for monitoring using following methods:

- processes having pattern "/apache/v2.2/bin/httpd" in MonitorProcessPatterns attribute
- a socket listening at host (isv2) and port (9191) specified as part of ListenAddressPort attribute.

In this example, the default monitoring sequence is changed using MonitorSequence attribute. First, a process check is done using MonitorProcessPatterns attribute and then socket connection test is done using ListenAddressPort attribute.

```
ABApache_zones abapache_res (
                StartProgram = "/apache/v2.2/bin/httpd -k start -f
/apache/v2.2/conf/httpd.conf"
                StopProgram = "/apache/v2.2/bin/httpd -k stop -f
/apache/v2.2/conf/httpd.conf"
                ListenAddressPort = "isv2:9191"
               MonitorProcessPatterns = { "/apache/v2.2/bin/httpd" }
                MonitorSequence = "MonitorProcessPatterns
ListenAddressPort PidFilesPatterns MonitorProgram"
                )
```

Example 4

In this example, a new agent ABApache_vcsone is created for base product VCS One and it is deployed on system sun-1.

```
# ./agentbuilder ABApache_vcsone -base vcsone -platform solaris
-system sun-1
Agent [ABApache_vcsone] created successfully!
Using [ssh] as default to deploy agent on [sun-1]
root@sun-1's password:
Successfully deployed agent on system : [sun-1]
```

Removing Agent Builder in a VCS environment

You can choose to uninstall Agent Builder after you have finished creating the agents. Perform the following steps to remove Agent Builder.

To remove Agent Builder

1. Log in as superuser.
2. Execute the following command:

Platform	Command
AIX	`# installp -u VRTSappab.rte`
HP-UX	`# swremove VRTSappab`
Linux	`# rpm -e VRTSappab`
Solaris	`# pkgrm VRTSappab`

Removing Agent Builder in a VCS One environment

You can remove Agent Builder using the installagpack program.

To remove Agent Builder

1. Mount the Agent Pack software disc on the Policy Master system on which you plan to run the uninstallagpack program.
2. Depending on the platform type, navigate to the directory containing the uninstaller for the VCS One agents:

AIX	`# cd aix/high_availability_agents`
HP-UX	`# cd hpux/hpux<os_version>\ /high_availability_agents`
Linux	`# cd linux/dist_arch/high_availability_agents` Where dist is the Linux distribution and arch is the architecture.
Solaris	`# cd solaris/dist_arch/\ high_availability_agents` where dist_arch is sparc or sol_x64

3. Start the uninstallagpack program.
   `# ./uninstallagpack -appab`
4. Enter the name of the Policy Master system.

5 Review the output as the program verifies the agent pack that you installed and removes the Agent Builder package.
You can view logs in the /var/VRTS/install/logs directory.

Removing the agent created by Agent Builder

Perform the following steps to remove the agent created using Agent Builder.

To remove the agent created by Agent Builder

1 Log in as superuser.

2 Remove all the resources of the resource type to be deleted.

3 Delete the resource type that was created using Agent Builder.

4 Remove the agent directory from all nodes in which the agent is deployed.

VCS 4.1	`# /opt/VRTSvcs/bin/`*`agent directory`*
VCS 5.0/VCS One 2.0	`# /opt/VRTSagents/ha/bin/`*`agent directory`*

Known Issue

For VCS version 4.x on AIX and HP-UX, the agent created by agentbuilder cannot be deployed to the specified system.

Workarounds

The following two workarounds are available:

- Option 1: Upgrade Perl to version 5.8.2 or later.
- Option 2: Though the newly created agent does not get deployed to the specified system, the agentbuilder utility creates the *`agentname.tar`* file. To deploy the newly created agent on each node:
  - Copy the *`agentname.tar`* file to the / directory.
  - Extract the *`agentname.tar`* file to get the agent files.