IBM PowerHA SystemMirror for AIX

Standard Edition

Version 7.2.1

Developing Smart Assist applications for PowerHA SystemMirror

IBM PowerHA SystemMirror for AIX

Standard Edition

Version 7.2.1

Developing Smart Assist applications for PowerHA SystemMirror

Note
Before using this information and the product it supports, read the information in "Notices" on page 65.

This edition applies to IBM PowerHA SystemMirror 7.2.1 Standard Edition for AIX and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this document v
Highlighting v
Case-sensitivity in AIX v
ISO 9000. v
Related information v

Developing Smart Assist applications for PowerHA SystemMirror 1
PowerHA SystemMirror Smart Assist development concepts 1
Concepts and terms used in Smart Assist development 1
Smart Assist requirements 2
Smart Assist framework 3
Overall flow of Smart Assist 3
Smart Assist identifiers and component identifiers . . 5
Packaging and installing 6
Developing SMIT panels 7
Smart Assist component discovery 9
Parameterized verification check files 14
Miscellaneous concepts and functionality . . . 16
Planning names and values for the Smart Assist discovery database 17
Smart Assist commands 18
Smart Assist registration and query 19
Smart Assist application registration and query 23
Convenience routines for SMIT panels 25
clvt API 25
Cluster class operations 25
Node class operations 27
Interface class operations 30
Network class operations 31
resource_group class 34
service_ip class 38
Application controller class use 40
application_monitor class 42
Resource group temporal_dependency class . . 45
Resource group location_dependency class . . . 46
file_collection class 47
Sample Smart Assist program 49
Overview 49
Installing the sample program 49
Deinstalling the sample program 50
Command to discover Smart Assist components 51
Add application instance functionality 51
Modify application instance functionality . . . 53
Deleting application instances 55
SMIT general application Smart Assist add stanzas 55
SMIT general application Smart Assist modify stanzas 59

Notices 65
Privacy policy considerations 67
Trademarks 67

Index 69

About this document

This document describes how you can develop a configuration and management tool that makes an installed application highly available by configuring PowerHA® SystemMirror®.

Highlighting

The following highlighting conventions are used in this document:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Case-sensitivity in AIX

Everything in the AIX® operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the **ls** command to list files. If you type `LS`, the system responds that the command `is not found`. Likewise, **FILEA**, **FiLea**, and **filea** are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Related information

- The PowerHA SystemMirror Version 7.2.1 PDF documents are available in the PowerHA SystemMirror 7.2.1 PDFs topic.
- The PowerHA SystemMirror Version 7.2.1 release notes are available in the PowerHA SystemMirror 7.2.1 release notes topic.

Developing Smart Assist applications for PowerHA SystemMirror

Use this information to develop a configuration and management tool that makes an installed application highly available by configuring PowerHA SystemMirror.

As a prerequisite for developing the tool, you should be familiar with:

- Planning and administering PowerHA SystemMirror, including an existing Smart Assist
- Communications, including the TCP/IP subsystem
- Coding for SMIT
- A scripting language such as Perl or KSH
- Knowledge of the target application.

PowerHA SystemMirror Smart Assist development concepts

Smart Assist is a tool to help end users configure PowerHA SystemMirror to make an instance of an application (such as a DB2® or Oracle instance, or a WebSphere® component) highly available.

Each Smart Assist manages the collection of PowerHA SystemMirror components needed to support a particular application. The user sees the collection of PowerHA SystemMirror components as a single entity, and in PowerHA SystemMirror that entity is represented by an application name. Smart Assists support adding, modifying and removing individual applications. When configuring PowerHA SystemMirror for new applications, the Smart Assist will ask the end user for the minimum amount of information necessary and then auto-detect the file systems, volume groups, service IP labels and other application resources used by the selected application instance. The Smart Assist then configures one or more PowerHA SystemMirror resource groups, application controllers and application monitors as needed to make the application highly available. This saves the end user steps in the configuration process and also helps ensure the proper configuration of the base application instance within PowerHA SystemMirror.

Smart Assists can be comprised of multiple components; each component supports a particular aspect of the target application. As an example, the Oracle Smart Assist has three components - one for supporting Oracle RDBMS, a second for supporting Oracle Application Server CFC, and a third for supporting Oracle Application Server AFC. Each of these components can have different Smart Assist properties, as well as different user interfaces within SMIT.

Smart Assists also integrate with cluster verification. Smart Assists can use the custom PowerHA SystemMirror verification methods to perform cluster verification. In the event a user modifies either the target application configuration, or the PowerHA SystemMirror configuration, custom verification routines will ensure the application will continue to function.

Concepts and terms used in Smart Assist development

Several concepts and their associated terms are used in the Smart Assist implementation.

- *Application*.

  Each Smart Assist supports an application. Smart Assists exist for DB2, WebSphere and Oracle. Also, a General Application Smart Assist (GASA) is packaged with PowerHA SystemMirror to support a generic application given user-supplied application start and stop scripts and related volume groups.
- *Application instance*.

When started, a Smart Assist usually detects an application instance previously created by the user. The DB2, WebSphere and Oracle Smart Assists all require a pre-existing application instance which must be set up by the user prior to using the respective Smart Assists.

- *Discovery of applications, application components, and application sub-features*.

  When the user selects the PowerHA SystemMirror SMIT **Add an Application to the PowerHA SystemMirror Configuration** menu a discovery script provided by each Smart Assist checks to see if the application is installed. If the application is installed, the discovery script then goes on to discover the particular supported component of the application, or sub-feature. The discovery script reports status back to the application discovery framework indicating whether the particular Smart Assist component can be used. Note that the use of the term "discovery" is different from the cluster discovery functionality in the PowerHA SystemMirror SMIT **Extended Configuration** menu.

- *Smart Assist application instance*.

  When the user selects an application instance to make highly available, the Smart Assist must create a Smart Assist application instance, which allows the Smart Assist framework to create the various PowerHA SystemMirror resource groups, applications and application monitors needed to make the base application instance highly available.

Smart Assist requirements

Smart Assist has several different requirements.

The Smart Assist is responsible for:

- Discovering the installation of the application and (if necessary) the currently configured resources such as service IP address, file systems and volume groups
- Providing a SMIT interface for getting configuration information from the user including a new Service IP address
- Providing a SMIT interface for changing PowerHA SystemMirror and application configuration information
- Defining an application to PowerHA SystemMirror and supplying custom start and stop scripts
- Supplying an application monitor for the application, if applicable
- Configuring a resource group to contain
  - Primary and takeover nodes
  - The application
  - The service IP address
  - Shared volume groups
- Configuring various resource group temporal and location dependencies, should the application solution require it
- Specifying files that need to be synchronized using the PowerHA SystemMirror File Collections feature
- Modifying previously configured applications
- Providing new verification methods
- Providing methods to test the application's cluster configuration (via the Cluster Test Tool), should the standard cluster test suite not suffice.

The Smart Assist framework described in this document is intended to be an easy-to-use framework for developing new Smart Assists. Some readers may find it helpful to first take a look at the example in Sample Smart Assist program and then return to the referenced sections when needed rather than reading this document from start to end.

Related concepts:

"Sample Smart Assist program" on page 49
These topics provides a sample Smart Assist program, based on the General Application Smart Assist (GASA).

Smart Assist framework

The Smart Assist Framework is the infrastructure PowerHA SystemMirror provides for the development of a PowerHA SystemMirror Smart Assist.

It includes:

- A discovery feature that automatically provides the end user a list of installed Smart Assists and the related applications (which may or may not be configured). If an application is not installed or is a different version than what the Smart Assist supports, then an appropriate message will be displayed.
- A way to enable a silent installation or configuration of software, if needed, prior to getting specific configuration information from the user through a SMIT panel.
- A Smart Assist API to allow the Smart Assist to change the cluster configuration and affect cluster operations. This is the interface for creating the PowerHA SystemMirror topology, resource group and resource configurations.
- A Smart Assist Registration API for saving and looking up information about an application configured using the Smart Assist.
- Utilities for installing and removing Smart Assist data as the fileset containing it is installed or removed from AIX. Any applications, resource groups, clusters, etc. configured with this Smart Assist will still remain, but will have to be maintained using the Standard and Extended PowerHA SystemMirror SMIT paths.
- A mechanism to test the resource group containing the applications with the Cluster Test Tool.
- A way to add custom verification methods to the cluster verification utility.
- A secure way to gather information from all nodes in the cluster.

Overall flow of Smart Assist

When Smart Assist is installed, the fileset or installation script adds the appropriate Smart Assist registry entries to the PowerHA SystemMirror as ODM using the framework's API **claddsa** and then inserts its SMIT screens into the SMIT ODMs. The registration ODM contains name / value pairs that tie the Smart Assist Identifier and component Identifier.

Note: If you are using a IBM® in house developed Smart Assist, the entries will be automatically added in with the help from the build system (packdep.mk file in the packages folder is used).

From the main Smart Assist SMIT **Add an Application to the PowerHA SystemMirror Configuration** menu, complete the following steps:

1. Check if both nodes are running latest framework for Smart Assists by looking at the value in the PowerHA SystemMirror ODM entry called SMARTASSIST_VERSION for GASA smart assist. If any of the nodes are running an old version go to Step 2. If the nodes are running the current version go to Step 3.
2. Each Smart Assist in the PowerHA SystemMirror ODM runs a custom script (called a discovery script) that detects if the application code supported by the Smart Assist is installed at the time. The Smart Assist menu then informs the end user the status of each Smart Assist (see below) and allows the user to select any Smart Assist that has the base application code installed. For Smart Assists whose discovery scripts do not detect installed instances of the target application, an entry will appear. However once selected, the user will have to navigate further than the initial discover screen.
3. Find out all the Smart Assists installed on both the nodes, and list them. Discovery scripts are not called at this time, and you should select a Smart Assist to proceed further (for ex. Smart Assist for SAP).
4. Select **Automatic Discovery and Configuration** or **Manual Configuration** for the type of configuration.

   Note: If you select **Automatic Discovery and Configuration** the discovery scripts runs for all the components available for the previously selected smart assist.

5. Select the component you want to work with. If there are more than one instance of the selected component that needs to be configured (for example, Different database instances in a given Oracle Database) you will have select the other instances. If there is only one instance that needs to be configured then it leads to a dialog screen.

Note: Note: If the user has already configured the cluster and nodes, there is no need to ask again for the communication path; the user would go directly from the **Make Applications Highly Available > Add an Application to the PowerHA SystemMirror Configuration** to the Selector screen.

```
*************************************************
 Configure PowerHA SystemMirror Cluster and Nodes

 Enter Communication Path to Nodes <Entry Fields>

*************************************************

 Select an Application from the List of Discovered...
 list of applications...

*************************************************
```

Note: Smart Assists with node names listed after their name are “active,” meaning that discovery detected instances of the application installed on the cluster nodes listed after the application type. In the figure above, no instances of DB2 or Oracle were discovered. When the Smart Assist is executed, it usually gets minimal information from the end user and then uses that information to detect any information it needs about the application instance it has to manage (file systems, volume groups, etc). It then uses the framework's configuration API to configure a PowerHA SystemMirror cluster and any necessary resource groups, applications and application monitors needed by the application instance. The Smart Assist framework used by the Smart Assist developer is:

- Several commands (User Interface API) used by the Smart Assist to add itself to the SMIT and HACMPsa ODM structures.
- A main Smart Assist SMIT menu (Make Applications Highly Available), which interfaces to the installed Smart Assists.
- Support for a discovery script, provided by the Smart Assist, and called from the SMIT **Add an Application to the PowerHA SystemMirror Configuration** menu, which detects whether the base application code is installed on a given node.
- A single command **clvt** (PowerHA SystemMirror Configuration API) gives access to 11 classes of objects representing the PowerHA SystemMirror components needed to make a base application instance highly available. The Smart Assist uses these commands to configure PowerHA SystemMirror to make the base application instance highly available.

The Smart Assist developer must use the framework, and develop a number of SMIT screens to create a Smart Assist for a particular application. There is a bit more to the Smart Assist framework but the above covers the basic functionality a Smart Assist developer must consider.

```
                    Make Applications Highly Available (Use Smart Assists)

Move cursor to desired item and press Enter

 Add an Application to the HACMP Configuration
 Change/Show an Application's HACMP Configuration
 Remove an Application from the HACMP Configuration

 Manage Your Applications
 Change/Show the Resources Associated with Your Application

 Test Your Application for Availability




F1=Help        F12=Refresh      F3=Cancel     F8=Image
F9=Shell       F10=Exit         Enter=Do
```

The **Make Applications Highly Available** SMIT menu lists the functionality the Smart Assist offers. Smart Assist Basics of Operation This section presents information on how a Smart Assist works in more detail. It discusses what issues you must consider and what code you need to write for your own Smart Assist.

Smart Assist identifiers and component identifiers

The Smart Assist Identifier and Component Identifiers uniquely identify a particular Smart Assist component within the Smart Assist framework.

The first step in developing a Smart Assist is defining these identifiers. The Smart Assist Identifier reflects the target application such as DB2, Oracle or WebSphere. Examples of Smart Assist IDs are "DB2_8.0" for DB2 and "Oracle_10G" for Oracle Application Server and RDBMS. Note that in these particular cases the Identifier also reflects the version of the application. To ensure version compatibility, it may be useful to develop separate Smart Assists for different versions of the target application. The Smart Assist Component Identifiers are used to subclassify a Smart Assist into the various subcomponents of an application, and into the sub-features of the Smart Assist. In the case of DB2 and Oracle, the database instances can be configured in different ways; therefore several Component Identifiers are needed, one for each way the database can be configured. Examples of DB2 Component Identifiers are shown in the table that follows:

Component Identifier	**DB2 Component Name (user visible)**
DB2_8.0_NON_DPF_SINGLE	DB2 Single Instance
DB2_8.0_NON_DPF_MUTUAL	DB2 Mutual Takeover

In the case of WebSphere, the Component Identifiers consists of several distinct functional components (a web server, a log server, etc.), and so each component has its own Component Identifier. Examples of WebSphere Component IDs are shown in the following table:

Component Identifier	WebSphere Component Name (user visible)
WAS_6.0_IHS_SERVER	IHS HTTP Server
WAS_6.0_APP_SERVER	WebSphere Application Server (Standalone)
WAS_6.0_DEPLOYMENT_MANAGER	WebSphere Deployment Manager
WAS_6.0_TRANSACTION_LOG_ RECOVERY	WebSphere Cluster Transaction Log

Packaging and installing

You can package your Smart Assist using tools appropriate to your environment.

The LPP package management system is the preferred method of packaging files on the AIX platform, but you can use RPMs, or other package and file management tools that provide the ability to:

- Install
- Remove
- Update
- Pre-require the PowerHA SystemMirror fileset **cluster.es.assist.common** to be installed prior to attempting to install the new Smart Assist fileset.

Adding a Smart Assist at install time

When Smart Assist is installed, it performs two tasks for each of its components.

These tasks are:

- Call the UI API routine **claddsa** which adds information about each component in the PowerHA SystemMirror ODM for later use by the framework. Some items specified at this time are the discovery script path and the Add and Modify SMIT menu paths.
- Add the particular Add and Modify SMIT screens specified for each component to the SMIT ODM for use when the end user invokes the Smart Assist. Any other dependent SMIT menus must be added to the ODM at this time as well.

Here is an example of a **claddsa** call:

```
claddsa -s "Apache_2" -c "APACHE_2.0_HS_SSL" \
   COMPONENT_ID="APACHE_2.0_HS_SSL" \
   SMARTASSIST_VERSION="1.0" \
   SMIT_ADD="clsa_apache_add" \
   SMIT_MODIFY="clsa_apache_modify \
   SMIT_ADD_TYPE="d" \ SMIT_MODIFY_TYPE="d" \
   DISCOVERY_COMMAND="/usr/es/sbin/cluster/sa/apache/sbin/discover"
\
   DEINSTALLATION_COMMAND="/usr/es/sbin/cluster/sa/apache/sbin/discover"
\
   REGISTRATION_COMMAND="/usr/es/sbin/cluster/sa/apache/sbin/register"
\
   DEREGISTRATION_COMMAND="/usr/es/sbin/cluster/sa/apache/sbin/deregister"
\
   MIGRATION_COMMAND="/usr/es/sbin/cluster/sa/sample/sbin/migrate"
\
   SA_ROOT="/usr/es/sbin/cluster/sa/sample/sbin/" \
   SA_NAME="Sample Smart Assist" \
   COMPONENT_NAME="Sample Smart Assist Component"
```

Related concepts:

"Smart Assist commands" on page 18
Use these topics as reference for the commands you use to develop a Smart Assist for PowerHA SystemMirror. Each topic lists syntax diagrams and provides examples for using each command.

Directory structure

Smart Assists should adhere to the directory structure underneath the Smart Assist name.

```
/usr/es/sbin/cluster/sa/SmartAssistID
```

Note: The SmartAssistID is the same Identifier used in the **claddsa** command.

Directory	Contents
./sbin/	Executables, and scripts that need to be executed on behalf of the Smart Assist; this includes any scripts run using the SMIT interface
./appserver/	PowerHA SystemMirror application controller start and stop scripts
./monitor/	PowerHA SystemMirror custom application monitors
./verification/	PowerHA SystemMirror cluster custom verification files for performing validation of configured applications
./smit	A collection of SMIT menus to be added to the PowerHA SystemMirror cluster. If the Smart Assist developer does not provide a mechanism at install time to add these ODM stanzas, it is required that the framework be able to find the SMIT ODM stanzas in this directory. One or more SMIT ODM stanzas may reside in this directory with the file extension 'odm'.

Developing SMIT panels

The Smart Assist you develop must fit into the current PowerHA SystemMirror SMIT panels designed for Smart Assists.

The SMIT menus are either installed via the Smart Assist at the time of installation, or alternatively the Smart Assist can request that the registration API inject the ODM stanzas. The preferred approach is to request the registration API to add the ODM stanzas, so that the Smart Assist deinstallation process can remove the ODM entries on your behalf.

The four SMIT stanza types: sm_cmd_opt, sm_cmd_hdr, sm_menu_opt, sm_name_hdr are supported by the PowerHA SystemMirror registration API. The Smart Assist framework keeps track of the entries added into the SMIT ODM stanzas, and at the time of deinstallation removes the appropriate entries simply by calling the deregistration command. If the Smart Assist performs the operation of adding the entries, then it is assumed by the framework that the Smart Assist will also remove the entries upon deinstallation.

There are several points in the Smart Assist framework where control will transfer from the Smart Assist framework to your new Smart Assist. You must use the correct Smart Assist Identifiers for the SMIT ODM classes in order for control to properly transfer from one session to another.

Related information:

System Management Interface Tool (SMIT)

Add an application to the PowerHA SystemMirror configuration

The **Add an Application to the PowerHA SystemMirror Configuration** SMIT session goes through a number of transitions before finally passing control off to the Smart Assist.

The user must first specify nodes for the cluster (if nodes exist). Next, you are presented a selector screen showing the available applications. Once the user selects an application, control is passed along to the Smart Assist Add screen.

If nodes and or sites were specified by the user, then the discovery information within SMIT would provide the following name value pairs:

```
#nodes: SmartAssistID_ComponentID
nodeA nodeB:<SmartAssistID_ComponentID>
```

Change or show an application's PowerHA SystemMirror configuration

Similar to the **Add An Application to the PowerHA SystemMirror Configuration** SMIT screen, users will select a particular application already defined to PowerHA SystemMirror, and then modify the existing values.

Control will be passed off to the Smart Assist once the user selects the application to change or show.

Manage your applications

The **Manage Your Applications** SMIT screen is different from the Add and Change/Show SMIT screens in that it transfers control over to a SMIT menu system written by the Smart Assist, rather than to a SMIT dialog.

After selecting the **Manage Your Applications** SMIT screen, the user is presented a list of applications that are already configured and have management screens. Once the user selects a particular component of the Smart Assist, control is passed to the Smart Assist menus. The Smart Assist developer must provide a sm_menu_opt SMIT ODM with an id of clsa-manage <your_next_id>.

General guidelines for developing SMIT

Each Smart Assist will need two developing SMIT displays at a minimum: One for adding an application and one for modifying an existing configuration. It is beyond the scope of this document to explain how to develop SMIT, but we can mention some requirements that facilitate a smooth flow from the Smart Assist menus to the displays that add or modify the application's configuration within PowerHA SystemMirror. The Smart Assist Framework will also supply some useful information you can choose to use.

You may find that you will need additional selectors prior to navigating to the SMIT display to add your application. For instance, if you are trying to make a database instance highly available, your Smart Assist may need to have the user select a specific database instance from a list. In this case, you may choose to have a selector rather than a display as an entry point. SMIT menus are not an option (due to SMIT requirements). To register the entry points for Add and Modify, when the Smart Assist is installed, the **clquerysa** command that needs to run must specify these parameters:

SMIT_ADD	**String containing the ID for the SMIT stanza.**
SMIT_ADD_TYPE	String "d" for display, "n" for selector. "d" by default.
SMIT_MODIFY	String containing the ID for the SMIT stanza.
SMIT_MODIFY_TYPE	String "d" for display, "n" for selector. "d" by default.

The Smart Assist Framework will provide values for the following cooked field names for use by the SMIT display or selector:

sa_id	**Smart Assist ID**
component_id	Component ID
cluster_name	Name of the cluster
nodes	Space-separated list of nodes on which the component was installed.
application_id	(for modify only) the application Identifier

The Smart Assist may need to offer a way of getting additional information from the user, or give the user operations to perform other than add, change/show, remove, and test.

You as the developer can provide additional SMIT menus, selectors, and dialogs under the **Manage Your Applications** menu heading of SMIT (clsa_manage). Your first sm_menu_opt stanza will need to start with a sequence number that does not collide with any of the existing Smart Assist sequence numbers. The fileset packaging for your Assist should check the available SMIT IDs and use the next available ID,

or pre-assign a set of IDs and check to ensure that range is available. If the ID is not available, the SMIT sm_menu_opt IDs will need to be changed by your installation script.

Note: As of this writing, sequence numbers of 100 and below and 900 and above are reserved for clsa_manage.

Smart Assist component discovery

When you select the main Smart Assist **Add an Application to the PowerHA SystemMirror Configuration** screen, the discovery scripts provided by each added Smart Assist component are called.

Each script checks to see if the base application (such as DB2 or WebSphere) is installed on the nodes of the cluster, and then checks to ensure the sub-component or sub-feature can be used on the local node where the discovery command is executed. If the functionality or application is installed, the Smart Assist is ”enabled“ on the menu and the node where the feature or application sub-component is accessible will be listed next to the Smart Assist. This gives the user a menu constructed in real-time indicating what applications are installed, and which Smart Assists are usable on which cluster nodes. If the application is not installed, the user can install it and restart the Smart Assist.

The discovery script is run on each node where the Smart Assist is installed; it is up to the Smart Assist developer whether to require the base application be installed on all defined cluster nodes. To ensure that multiple nodes, or all cluster nodes contain the Smart Assist fileset, you can use the parameterized verification functionality. This functionality is described in the section Parameterized verification check files.

Related reference:

“Parameterized verification check files” on page 14
Each parameterized verification check file can contain one or more different parameterized verification checks.

Example: Smart Assist discovery script

Here is an example of a Smart Assist discovery script.

```
#!/bin/ksh93
#
# Apache 2.0 Discovery script
#
# This script will determine if Apache is installed, and if so, if the
#mod_ssl.so module is installed on the local node. If both conditions
#are true, the script will output the following string:
#
# Apache v2.0 Smart Assist:APACHE_2.0:Hot-Standby SSL Apache
#Server:APACHE_2.0_HS_SSL:1
#
# Otherwise, the last 2 characters will be replaced with :0 noting that
#the apache/ssl package is not properly installed on the local node.
#
dspmsg -s 1 apache.cat 1 "Apache v2.0 Smart Assist"
print -n ":APACHE_2.0:"
dspmsg -s 1 apache.cat 2
"Hot-Standby SSL Apache Server"
print -n ":APACHE_2.0_HS_SSL:"
# Simple example of scanning for the apache RPM on an AIX machine
result=$(rpm -qa 2>/dev/null | grep apache)
[[ -z $result ]] && {
 echo "0"
 exit 0
}
# If SSL is installed in this version of apache, this node has SSL
#available
rpm -ql apache 2>/dev/null | grep mod_ssl.so >/dev/null
```

```
 (( $? == 0 )) && {
 echo "1"
}
echo "0"
```

Invoking the Smart Assist's component Add menu

If you choose a Smart Assist with an application installed, the framework then displays a list of component names.

These names are generated from the component discovery process. You must select a component and the SMIT Add menu specified when the Smart Assist was added is invoked. The code in the SMIT menu generated by the Smart Assist developer is then executed.

The Smart Assist framework uses the two name value pairs sent to **claddsa**: SMIT_ADD and SMIT_ADD_TYPE. The value for SMIT_ADD_TYPE is either 'n' or 'd'. These values correspond to the values used in smitty ODM classes such as sm_name_hdr, in particular the next_type field.

Specifying 'n' means that after you select the particular Smart Assist component, the next SMIT ID will be a sm_name_hdr entry. In most cases the sm_name_hdr SMIT screen is used to select a particular application instance (for example, a DB2 instance name) or option before entering the ADD dialog. Specifying 'd' for SMIT_ADD_TYPE means that the next SMIT identifier will be a sm_cmd_hdr entry (dialog). The SMIT_ADD field for the Smart Assist component designates the next identifier SMIT will visit.

Before the particular Smart Assist SMIT stanzas are invoked, the application discovery framework builds the following cooked name fields. These fields can be used in SMIT for cmd_to_classify, cmd_to_discover, or cmd_to_exec functions.

Once control is passed within SMIT to the particular Smart Assist no further application framework SMIT screens are invoked. The developer is free to construct the sm_name_hdr, and sm_cmd_hdr entries as necessary.

Cooked name in SMIT	Description
sa_id	Smart Assist ID
component_id	Component ID
cluster_name	Name of the cluster
nodes	Comma-separated list of nodes on which the component was installed.
application_id	(for the modify screen only) the application identifier

Typical Add menu functionality for a Smart Assist component:

Typically the application instance is already configured; the task is to use SMIT panels to get user input and detect the resources already used by the application instance, and make them known to PowerHA SystemMirror so they can be made highly available.

You are asked to enter instance names. Then volume group names, file system name, and service labels are usually determined from that input.

The Smart Assist application instance is then created using the **claddsaapp** API routine, which stores instance information in the HACMPsa_metadata ODM. Other attributes of the Smart Assist application instance can be registered as needed.

```
/usr/es/sbin/cluster/sa/sbin/claddsaapp -a example_app \
 APPLICATION_NAME=" example_app" \
 RESOURCE_GROUP="example_app_group" \
 SMARTASSIST_ID="zzOther" \
 COMPONENT_ID="GASA"
```

The cluster configuration API (**clvt**) is then used by the Smart Assist to configure the resource groups and application monitors needed by the application.

The very first call the add script makes should be to the application framework API **/usr/es/sbin/cluster/sa/sbin/clsapre** . This script can be passed the **-c** flag to change the name of the cluster to suit the particular application being configured if needed.

After the add script completes successfully, the developer should call the script **/usr/es/sbin/sa/sbin/clsapost -v.** Note that the **-v** flag will run verification and synchronization as the last step of adding a application instance.

See the GASA script **/usr/es/sbin/cluster/sa/gasa/sbin/add** as an example of the add script.

Related concepts:

"clvt API" on page 25
These topics describe the **clvt** (cluster virtualization tool) API.

Change or show existing Smart Assist application instance with PowerHA SystemMirror resources:

The **Change/Show an Application's PowerHA SystemMirror Configuration** menu can provide you a way to make minor changes to a given PowerHA SystemMirror configuration.

When selected, the **Change/Show an Application's PowerHA SystemMirror Configuration** displays a list of Smart Assist application instances that can be changed. The Smart Assist developer must provide the screens to make the desired changes. Again, changes are made using the cluster configuration API **clvt**.

```
               Make Applications Highly Available (Use Smart Assists)

Move cursor to desired item and press Enter

 Add an Application to the PowerHA SysteMirror Configuration
 Change/Show an Application's PowerHA SysteMirror Configuration
 Remove an Application from the PowerHA SysteMirror Configuration

 Manage Your Applications
 Change/Show the Resources Associated with Your Application

 Test Your Application for Availability
  ┌──────────────────────────────────────────────────────────────────────┐
  │   Select an Application from the List of Discovered Applications Below   │
  │ Move cursor to desired item and press Enter.                         │
  │  example_app                                                         │
  │ F1=Help        F12=Refresh      F3=Cancel     F8=Image               │
  │ F9=Shell       F10=Exit         Enter=Do                             │
  │ /=Find         n=Find Next                                           │
  └──────────────────────────────────────────────────────────────────────┘
```

See the GASA script **/usr/es/sbin/cluster/sa/gasa/sbin/modify** as an example of a Modify change/show script.

Once you select a particular application instance, (example_app in the figure above) the application framework reads the SMIT_MODIFY_TYPE and SMIT_MODIFY fields from the HACMPsa ODM. These entries are defined when the Smart Assist fileset, or package is installed, and the installer calls the **claddsa** Smart Assist command.

```
                    General Applications Smart Assist

Type or select values in entry fields
Press Enter AFTER making all desired changes

                                       [Entry Fields]
 * Application Kane                    example_app
 * Primary Node                        [cujo]                      +
 * Takeover Nodes                      [cemetery]                  +
 * Application Controller Start Script     [/exampleapp/start]
 * Application Controller Stop Script      [exampleapp/stop]
 * Service IP Label                    [service1]                  +




F1=Help        F12=Refresh      F3=Cancel     F4=List
F5=Reset       F6=Command       F7=Edit       F8=Image
F9=Shell       F10=Exit         Enter=Do
```

The Smart Assist Manage menu:

As a SMIT limitation, Add and Change/Show menus can only be displays and selectors. You as the developer can provide additional SMIT menus, selectors, and dialogs under the **Manage Your Applications** menu heading of SMIT (clsa_manage).

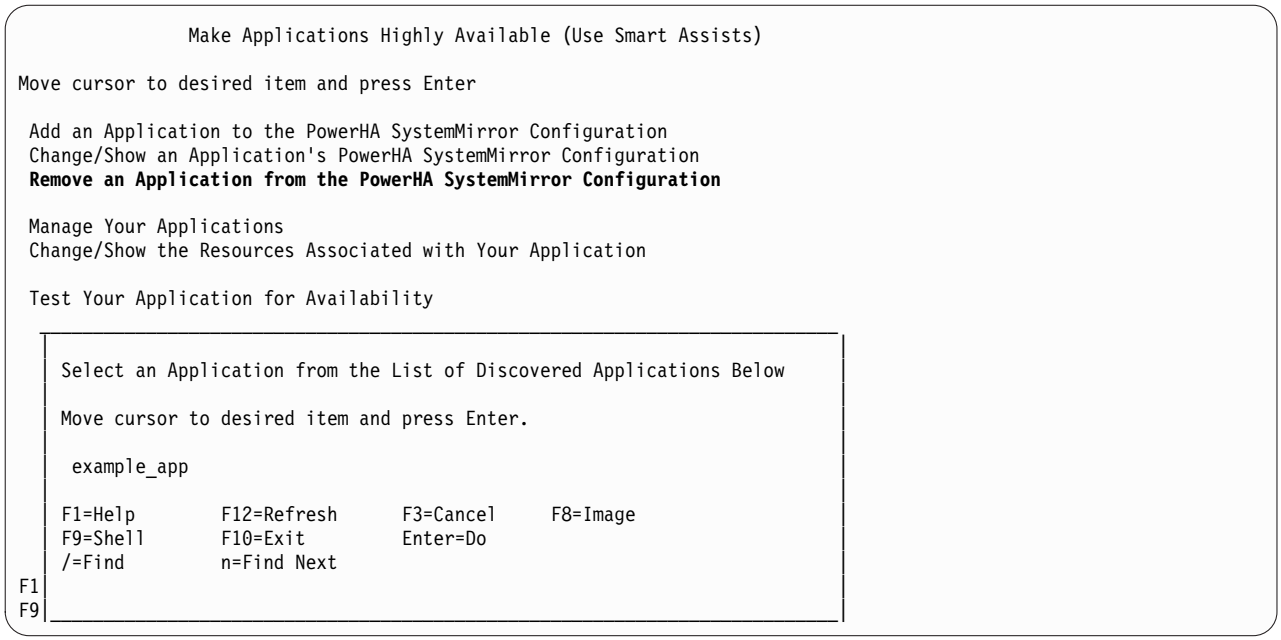
Your first sm_menu_opt stanza will need to have clsa_manage with a sequence number that does not collide with other sequence numbers. You will need to install the SMIT ODM stanzas at install time through **odmadd** calls within the install script. None of the cooked field names mentioned previously will be available for use by SMIT.

Note: Sequence numbers of 100 and below and 900 and above are reserved for clsa_manage screens developed by the Smart Assist developer.

Removing Smart Assist application instances:

The **Remove an Application from the PowerHA SystemMirror Configuration** menu selection presents a list of Smart Assist application instances.

To remove a Smart Assist application instance, select the one you want to delete and the **clrmsaapp** UI API routine removes the instance data from HACMPsa_metadata. Any resource groups created are typically kept so they can be managed by the user using the regular PowerHA SystemMirror functionality.



Any DEREGISTRATION_COMMAND specified in the **claddsa** call made by the Smart Assist is called before the above actions are taken.

Removing Smart Assists:

Removing a Smart Assist is the reverse of the **Add a Smart Assist at Install Time** operation.

It is done by an uninstall script:

- The Add, Change/Show and Manage SMIT menus must be removed from the SMIT ODM.
- The entries in the Smart Assist ODMs are removed using the **clrmsa** UI API routine.

Any resource groups created are typically kept so they can be managed by the user using the regular PowerHA SystemMirror functionality.

Custom verification and parameterized verification checks

Once the cluster is configured via the Smart Assist add or modify script, the final step is a verify and synchronize, which is performed via the **clsapost -v** script.

At that time, a number of checks can optionally be made on the nodes in the cluster to make sure the necessary resources exist.

Developers of Smart Assists have two methods of introducing new verification checks into the PowerHA SystemMirror product:

1. Custom verification methods. This is the preexisting PowerHA SystemMirror mechanism used to perform third-party verification for system or application components that are not using PowerHA SystemMirror.
2. Parameterized verification checks, described in this section.

Parameterized verification checks are described files that reside in the **/usr/es/sbin/cluster/etc/config/verify** directory.

These files have an extension of**.ver** and are typically created when the Smart Assist instance is created via the REGISTRATION_COMMAND. They make specific checks to ensure that the resources needed by the instance exist and are sufficient. The checks that can be made are:

- APAR is loaded
- Disk space is available
- File exists
- Fileset is installed
- Group exists
- Swap space exists
- User exists.

Related reference:

"Parameterized verification check files"
Each parameterized verification check file can contain one or more different parameterized verification checks.

Parameterized verification check files

Each parameterized verification check file can contain one or more different parameterized verification checks.

Each file has the following attributes which apply globally to all parameterized verification checks within the file. Add comments to the file by starting the line with the pound sign (#).

```
Component.Name.DefaultName = "Default name as visible in error or
warning messages"
Component.MsgCat.ID = 1
Component.MsgCat.Set = 10
Component.MsgCat.Catalog = "myassist.cat"

Component.Nodes = "AlL"
or
Component.Nodes = "S=<smart assist ID>:A=<ApplicationID>"
or
Component.Nodes = "LOCALNODE"

Component.Name.DefaultName
```

Component.Name.DefaultName is the name that is used in verification errors or warnings produced by the parameterized verification checks listed below.

Component.MsgCat

- ID - message catalog identifier as used in dspmsg
- Set - The message catalog set as used in dspmsg
- Catalog - the message catalog name as used in dspmsg

Component.Nodes

- ALL - Verify on all available nodes
- LOCALNODE - Verify only on the local node
- S=<SmartAssistID>:A=<ApplicationID> Verifies the nodes associated with the specific application instance.

APAR verification

The APAR parameterized verification check validates that the set of APARs specified is installed on the nodes defined by *Component.Nodes.*

If one or more nodes are missing the required fileset, a verification message is printed, the severity of the message (error or warning) is determined by the value of *HAVerify.APAR.severity,* which can be either "ERROR" or "WARNING" as shown below:

```
HAVerify.APAR.severity = "ERROR | WARNING"
HAVerify.APAR.exists[0].apar = "IY7265H"
HAVerify.APAR.exists[1].apar = "IY72657"
.
.
.
HAVerify.APAR.exists[n].apar = "..."
```

Disk space verification

The disk space parameterized verification check validates that the specified file systems have sufficient space (as specified).

The format of the diskspace verification check is as follows:

```
HAVerify.DiskSpace.severity = "ERROR | WARNING"

#Validate /var filesystem has 200 MB of free space
HAVerify.DiskSpace.check[0].filesystem = "/var"
HAVerify.DiskSpace.check[0].minsize = "200MB"

# Validate /oradata has 1GB of free space
HAVerify.DiskSpace.check[1].filesystem = "/var"
HAVerify.DiskSpace.check[1].minsize = "200GB"

# Validate /orasoft has 4000Kb of free space
HAVerify.DiskSpace.check[2].filesystem = "/orasoft"
HAVerify.DiskSpace.check[2].minsize = "4000Kb"
...
```

The qualifiers for minsize are # followed by [Kb - KiloBytes | MB - MegaBytes | GB - GigaBytes | B - Bytes].

File verification

The file verification determines if the set of specified files exist on the specified nodes.

If not, an error or warning message is produced in the verification output. The format of the File section is as follows:

```
HAVerify.File.severity = "ERROR | WARNING"
HAVerify.File.exists[0].name = "/etc/hosts"
HAVerify.File.exists[1].name = "/orasoft/10g/admin/dbs/asdb/pfile/
warehouse_pfile.ora"
.
.
.
HAVerify.File.exists[n].name = "..."
```

Fileset verification

The fileset parameterized verification check validates that the set of LPP filesets is installed on the set of nodes present in the header information of the file.

The format of the fileset section follows:

```
HAVerify.Fileset.severity = "ERROR | WARNING"

# Detect cluster.es.server.cfgast version installed
HAVerify.Fileset.exists[0].name = "cluster.es.server.cfgast"
HAVerify.Fileset.exists[0].version = "n.n"

# Detect cluster.es.server.testtool is installed (no version required)
HAVerify.Fileset.exists[1].name = "cluster.es.server.testtool"
```

Group verification

The group validation parameterized verification check validates that the AIX group is defined on the set of nodes specified in the header with an ID as specified in the section.

One or more groups can be validated. The format of the section follows:

```
HAVerify.Group.exists.severity = "ERROR | WARNING"
HAVerify.Group.exists[0].name = "dba"
HAVerify.Group.exists[0].GID = 100
```

User verification

The user validation parameterized verification check validates that the AIX user is defined on the set of nodes specified in the header.

The user must have the same ID as specified in the section. One or more users can be validated within one verification script. The format of the section follows:

```
HAVerify.User.exists.severity. = "ERROR | WARNING"
HAVerify.User.exists[0].name = "oracle"
HAVerify.User.exists[0].UID = 100 ...
```

Swap space verification

The swap space parameterized verification check validates that the swap space for the set of nodes specified in the header meets the requirements for free swap space and total available swap space.

The format of this section is as follows:

```
HAVerify.SwapSpace.severity = "ERROR | WARNING"
# At a minimum 1024MB must be allocated to the swap space
HAVerify.SwapSpace.minsize = "1024MB"

# At a minimum 512MB of space must be available at the time verification
executes
HAVerify.SwapSpace.minfree = "512MB"
```

Miscellaneous concepts and functionality

This section describes additional functionality available to the Smart Assist developer.

Define the cluster automatically

If the cluster is not defined when the **Add an Application to the PowerHA SystemMirror Configuration** menu selection is made, the Smart Assist framework automatically queries the end user for the nodes in the cluster and creates the cluster. This step is equivalent to the **Configure a PowerHA SystemMirror Cluster and Nodes** SMIT function.

Query and list commands used in the API

Several utility commands are available to the Smart Assist developer. See Smart Assist commands for a detailed description of these routines:

- **cllsssaapp** - List configured Smart Assists application instances.
- **cllsserviceips** - List available service IP labels
- **clquerysa** - Return attributes of specified Smart Assist
- **clquerysaapp** - Return attributes of specified Smart Assist application instance

The **clsapre** and **clsapost** routines must be called before and after creating a Smart Assist application instance. They provide hooks back to the framework during these operations.

```
clsapre [ -c cluster_name ]
-c cluster_name Change the name of the cluster to the specified value
clsapost [ -v ]
-v Run cluster verification after the application instance is added
```

Registration and deregistration commands

The REGISTRATION_COMMAND and DEREGISTRATTION_COMMAND specified in the **claddsa** call allow the Smart Assist developer to perform activities before the Smart Assist is installed and before it is removed.

Smart Assist migration commands

The MIGRATION_COMMAND specified in the **claddsa** call allows the Smart Assist developer to specify a migration script in case a newer Smart Assist is superseding an older one. If the SMARTASSIST_VERSION values don't match when the **claddsa** call is made, the MIGRATION_COMMAND specified in the new Smart Assist is called to migrate any existing Smart Assist application instance data to a new format if necessary, or to make modifications to the existing data as necessary.

Smart Assist test command

If a TEST_COMMAND is specified in the **claddsa** call, the script is called when the end user selects the **Test Your Application for Availability** menu selection. The test script will be given the name of the application to test as the first argument. It is the test script's responsibility to produced stdout output that conforms to the cluster test tool script format. If a TEST_COMMAND is not specified a default test script will execute if the user chooses to test Planning Names and Values for the Smart Assist for the Discovery Database.

Related concepts:

"Smart Assist commands" on page 18
Use these topics as reference for the commands you use to develop a Smart Assist for PowerHA SystemMirror. Each topic lists syntax diagrams and provides examples for using each command.

Planning names and values for the Smart Assist discovery database

When a Smart Assist is installed using **installp** or the SMIT equivalent, the Smart Assist must add information about itself to the discovery database, PowerHA SystemMirror, in order to ensure that the discovery process will find the applications that can be made highly available.

The installation script associated with the fileset must register each component of the Smart Assist using the **claddsa** command. The information about each Smart Assist should be stored in an ODM or similar structure so that cluster synchronization will distribute this information to the rest of the cluster. Each component of a Smart Assist should specify some or all of the following information. This information is needed for discovery and other processes:

Entry	Type	Required	Default Value	Notes
Smart Assist ID	String - 32 characters or less	yes		Must be common to all components of the Smart Assist
Component ID	String -32 characters or less	yes		Must be unique among the components of the Smart Assist
Smart Assist Version	Number	yes		Must be greater than previous versions
Discovery Command	Full File Path plus arguments	yes		See the description and requirements for the Discovery Script.

Entry	Type	Required	Default Value	Notes
Uninstallation Command	Full File Path plus arguments	no	none	Used if special steps must be taken to completely uninstall a Smart Assist.
Registration Command	Full File Path plus arguments	no	none	The optional command is run before the Add SMIT path is invoked
Deregistration Command	Full File Path plus arguments	no	none	The optional script is run before the resource group and resources have been removed from PowerHA SystemMirror and the information removed from the Smart Assist databases.
Migration Script	Full File Path plus arguments	no	none	This optional script is run if **claddsa** detects a previous version was installed.
SMIT Add Screen or Selector	SMIT Path			
SMIT Add Type				
SMIT Modify Screen or Selector	SMIT Path			
SMIT Modify Type				
Resource Group Name Generation Script	Full File Path plus arguments	Yes		Used by the Registration Process
Application Sontroller Name Generation Script	Full File Path plus arguments	yes		Used by the Registration Process
Custom Test Script	Full File path	no	Uses the Application-based utility for generating a test plan	

Smart Assist commands

Use these topics as reference for the commands you use to develop a Smart Assist for PowerHA SystemMirror. Each topic lists syntax diagrams and provides examples for using each command.

Highlighting

The following highlighting conventions are used in this topic:

Item	Description
Bold	Identifies command words, keywords, files, directories, and other items whose actual names are predefined by the system.
Italics	Identifies parameters whose actual names or values are supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Reading syntax diagrams

Usually, a command follows this syntax:

Item	Description
[]	Material within brackets is optional.
{}	Material within braces is required.
|	Indicates an alternative. Only one of the options can be chosen.
...	Indicates that one or more of the kinds of parameters or objects preceding the ellipsis can be entered.

Related information

For complete information on a command's capabilities and restrictions, see the online man page. Man pages for PowerHA SystemMirror for AIX commands and utilities are installed in the **/usr/share/man/cat1** directory. Use the following syntax to read man page information where *command-name* is the actual name of the PowerHA SystemMirror command or script.:

```
man command-name
```

For example, type **man clpasswd** to obtain information about the PowerHA SystemMirror user password command.

Related concepts:

"clvt API" on page 25
These topics describe the **clvt** (cluster virtualization tool) API.

Smart Assist registration and query

These commands add or remove a Smart Assist from the discovery and registration databases.

claddsa command

Registers a Smart Assist for use in application discovery.

Syntax

```
claddsa -s SmartAssistID -c ComponentID [-C] name1= "value1"
name2=name1= "value1"..
```

Parameters

-s *SmartAssistID*	**Unique Identifier for the Smart Assist**
-c *ComponentID*	Unique Identifier for the Smart Assist Component
`name1= "value1"...` `nameN= "valueN"`	Names of the name value pairs to be stored within the database.
-C	Causes a check to see if the Smart Assist and component can be configured on the local node (software is installed).

If you specify a Migration Script, then **claddsa** will first check to see if a previous version of the Smart Assist was installed. After the database is updated with the values specified on the command line, then the Migration script will be invoked, passing in the version of the previous Smart Assist via the CLSA_VER environment variable. Please note that unlike other values, the MIGRATION_COMMAND will not be stored in the HACMPsa ODM. The following list details the name and value pairs that the Smart Assist framework recognizes. All of these require both the Smart Assist identifier and the component identifier to be Smart Assist wide. Note: Please check to ensure that the Smart Assist and component identifiers used are unique.

Name	Description
SMARTASSIST_ID	Unique identifier for the Smart Assist
COMPONENT_ID	Unique string to identify the component within the Smart Assist
SMARTASSIST_VERSION	Unique string that identifies the Smart Assist version. This is used to determine if migration is necessary.
MIGRATION_COMMAND	In the event an older version of the same Smart Assist is already installed, the new migration script will run before the older version is removed. The script should migrate any existing application instances defined to PowerHA SystemMirror to the new Smart Assist format. Note: unlike other values, the value for this entry will not be stored in the HACMPsa ODM.
SMIT_MODIFY_TYPE	Either 'd' for sm_cmd_hdr screens, or 'n' for sm_name_hdr screens
SMIT_MODIFY	SMIT identifier used to navigate to either the name selector screen, or the cmd hdr dialog screen when modifying the particular Smart Assist component
DISCOVERY_ COMMAND	Command to discover whether the particular application component or application sub-feature is accessible on the local cluster node
DEINSTALLATION_ COMMAND	Command that executes before the Smart Assist is removed from the local node
REGISTRATION_ COMMAND	The command that executes when the user adds an application instance to the PowerHA SystemMirror configuration
DEREGISTRATION_ COMMAND	Command that executes before the Smart Assist framework removes the application from the PowerHA SystemMirror configuration
SMIT_ADD_TYPE	Either 'd' for sm_cmd_hdr screens, or 'n' for sm_name_hdr screens
SMIT_ADD	SMIT identifier used to navigate to the Smart Assist add screen for the user selected Smart Assist component.

You can add additional Smart Assist component name value pairs as needed. The Smart Assist framework will not interpret any name and value pairs that are not one of the above. If **-C** is specified, then the last step is to run the DISCOVER_COMMAND (if specified). If the command returns information, then print out a message of the form:

```
The following components of <Smart Assist> can be configured on the
local node because the software is installed:
      <component 1>
      <component 2>
      ...
Please run "smit clsa" to start configuring them to make them highly
available with PowerHA SystemMirror.
```

claddsa example

```
claddsa -s MYSMARTASSIST_8.0 -c FIRST_COMPONENT \
SMARTASSIST_ID="MYAPP_8.0" \
COMPONENT_ID="FIRST_COMPONENT" \
DISCOVERY_COMMAND="/usr/es/sbin/cluster/sa/MyApp/sbin/discovery"\
DEINSTALLATION_COMMAND="/usr/es/sbin/cluster/sa/MyApp/install/uninstall"
\
REGISTRATION_COMMAND="/usr/es/sbin/cluster/sa/MyApp/sbin/register"\
DEREGISTRATION_COMMAND="/usr/es/sbin/cluster/sa/sbin/MyApp/deregister" \
SMARTASSIST_VERSION="1.0" \
SMIT_ADD_TYPE="n" \
SMIT_ADD="clsa_mysmartassist_add_selector"
SMIT_MODIFY_TYPE="d" \
SMIT_MODIFY="clsa_mysmartassist_modify_dialog"
```

The resulting HACMPsa ODM entries will be:

```
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"
  component_id ="FIRST_COMPONENT"
  name="SMARTASSIST_ID"
  value="MYSMARTASSIST_ID"
HACMPsa:
```

```
  sa_id ="MYSMARTASSIST_8.0"
  component_id ="FIRST_COMPONENT"
  name="COMPONENT_ID"
  value="FIRST_COMPONENT"
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"
  component_id ="FIRST_COMPONENT"
  name="DEREGISTRATION_COMMAND"
  value="/usr/es/sbin/cluster/sa/sbin/MyApp/deregister"
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"
  component_id ="FIRST_COMPONENT"
  name="SMARTASSIST_VERSION"
  value="1.0"
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"
  component_id ="FIRST_COMPONENT"
  name="SMIT_ADD_TYPE"
  value="n"
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"
  component_id ="FIRST_COMPONENT"
  name="SMIT_ADD"
  value=" clsa_mysmartassist_add_selector"
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"component_id ="FIRST_COMPONENT"
  name="SMIT_MODIFY_TYPE"
  value=" clsa_mysmartassist_add_selector"
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"
  component_id ="FIRST_COMPONENT"
  name="SMIT_MODIFY"
  value=" clsa_mysmartassist_add_selector"
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"
  component_id ="FIRST_COMPONENT"
  name="DISCOVERY_COMMAND"
  value="/usr/es/sbin/cluster/sa/MyApp/sbin/discovery"
HACMPsa:
  sa_id ="MYSMARTASSIST_8.0"
  component_id="FIRST_COMPONENT"
  name="REGISTRATION_COMMAND"
  value="/usr/es/sbin/cluster/sa/MyApp/sbin/register"
```

clquerysa command

This commands queries the nodes of the cluster for available Smart Assist components and whether the component can run on the remote node.

-t discover : Queries the nodes of the cluster for available Smart Assist components and whether the component can run on the remote node.

-t filter : Filters the raw list of Smart Assists [and or components] to produce a list usable by SMIT for selecting a Smart Assist.The raw list has to be passed in by standard input.

Syntax for discovery

```
clquerysa -t discover [-n nodename]
```

Parameters for discovery

Parameter	Description
-n *nodename*	If **-n** is specified, then only the information from the remote node is returned.

Note: `clquery -t discover` is equivalent to:

```
for each node in the cluster
   clquerysa -t discover -n $node
```

Discovery output

The output that results is one line of text for each installed component with the following information:

Item	Description
Node	Name of the node on which discovery found the component.
SmartAssistName	Internationalized name of the Smart Assist.
SmartAssistID	ID of the Smart Assist; it must match what is in the Discovery DB.
ComponentName	Internationalized name of the Smart Assist component.
ComponentID	ID of the Component; it must match what is in the Discovery DB.
[0|1]	0 if the component is not installed; 1 if it is installed.

Syntax for filter

```
clquerysa -t filter
```

Filters the raw list of Smart Assists and components to produce a list usable by SMIT for selecting a Smart Assist. Only the Smart Assists names and the relevant nodes will be displayed (selector shown after the user selects **Add an Application to the PowerHA SystemMirror Configuration**):

```
DB2 UDB non-DPF Smart Assistant
 Oracle Smart Assist #
WebSphere Smart Assistant #
Other Applications # NodeA NodeB

clquerysa -t filter -s SmartAssistID
```

Filters the raw list of Smart Assists and components to produce a list usable by SMIT for selecting a Smart Assist component. Only the available components of a given Smart Assist will be given, as in:

```
DB2_8.0
DB2 Hot Standby
DB2 Mutual Takeover

clquerysa -t filter -s SmartAssistID -c Component ID
```

Filters the raw list of Smart Assists and components so only the list of nodes for the application component of a Smart Assist will be displayed as a comma separated list.

clrmsa command

Removes all Smart Assist components or a specified component.

Syntax

clrmsa -s *SmartAssistID -c ComponentID*

Parameters

Parameter	Description
-s *SmartAssistID*	Unique Identifier for the Smart Assist to remove.
-c *componentID*	ID of component to remove

First, **clrmsa** runs the uninstallation command for all components (unless you specify one). Then it removes all information pertaining to the Smart Assist from the discovery and registration databases HACMPsa and HACMPsa_metadata. It does not remove the application from the PowerHA SystemMirror configuration; it removes only the associated Smart Assist configuration. This allows the user to continue to manage the application using the regular SMIT PowerHA SystemMirror menus.

clrmsa example

Remove the Smart Assist zzother: clrmsa -s zzOther

Smart Assist application registration and query

These commands add or remove a Smart Assist from the discovery and registration databases of applications.

claddsaapp command

Registers information about an application in the **HACMPsa** database.

Syntax

```
claddsaapp -a [ApplicationID ] name1="value1" name2="value2" ...
nameN="valueN"
```

Parameters

Parameter	Description
-a	Invoking the **claddsaapp** command with just the **-a** switch lists the next available application ID that can be used.
-a *ApplicationID*	Lists names of the applications included in the *name=value* pairs
name=value **pairs**	Names of application components.

Required name / value pairs for this command are the SMARTASSIST_ID, COMPONENT_ID, and APPLICATION_NAME.

Item	Description
SMARTASSIST_ID	Smart Assist identifier that this particular application instance implements.
COMPONENT_ID	Smart Assist component identifier.
APPLICATION_NAME	This is the same value as the value passed to the **-a** flag or the ApplicationID.

claddsaapp example

```
claddsaapp -a MyApp \
    SMARTASSIST_ID="zzOther" \
    COMPONENT_ID="GASA" \
    APPLICATION_NAME="MyApp" \
    RESOURCE_GROUP="MyApp_group"
```

clquerysaapp command

Returns information about a registered application.

Syntax

```
clquerysaapp -a applicationID name1 ... nameN
```

Parameters

Parameter	Description
-a *ApplicationID*	Lists names of the applications included in the *name=value* pairs
name=value **pairs**	Names of application components.

If a name is specified, then only values associated with that name are returned in the form of *name =VALUE* (if more than one value is present, they are comma-separated). If more than one name is specified, these are presented as *NAME1=VALUE1 ... NAMEN=VALUEN* Specifying no names means all information will be returned as if all applicable names were specified on the command line, but in no defined order.

clquerysaapp examples

Query all names/values for MyApp:

```
clquerysaapp -a MyApp
SMARTASSIST_ID="zzOther"
COMPONENT_ID="GASA"
RESOURCE_GROUP="MyApp_group"
APPLICATION_NAME="MyApp"
```

Query the name of the Smart Assist for MyApp:

```
clquerysaapp -a MyApp SMARTASSIST_ID
SMARTASSIST_ID="zzOther"
```

clrmsaapp command

Removes specific or all information about a registered Smart Assist.

Syntax

```
clrmsaapp -a applicationID
clrmsaapp -a applicationID [name1|name1=VALUE1] ... [nameN|nameN=VALUEN]
clrmsaapp -s SmartAssistID
```

Parameters

Parameter	Description
-a *ApplicationID*	Lists names of the applications included in the *name=value* pairs
name=value **pairs**	Names of application components.
-s	Removes information about all applications configured with the specific Smart Assist ID.

After a component is registered or further configured, the application must be verified and synchronized. This is to ensure the registration database is synchronized with all nodes of the cluster.

clrmsaapp examples

Remove all information about MyApp:

```
Clrmsaapp -a MyAPP
```

cllssaapp command

Returns a list of applications configured by a Smart Assist.

Syntax

```
cllsaapp
```

Example output

```
DB2_UDB_db2inst1
DB2_UDB_db2inst2
OracleAFC_ias10g
OracleCFC_ias10g.
```

Convenience routines for SMIT panels

These routines must be called before and after creating a Smart Assist application instance. They provide hooks back to the framework during these operations.

`clsapre [ -c cluster_name ]`

Parameter	Description
-c *clustername*	Change the name of the cluster to this value.

`clsapost [ -v ]`

Parameter	Description
-v	Run cluster verification after the application instances is added

---

clvt API

These topics describe the **clvt** (cluster virtualization tool) API.

The **clvt** commands perform PowerHA SystemMirror cluster operations. Cluster operations include adding, deleting, and querying cluster objects, as well as bringing nodes on or offline and modifying resource groups and interfaces.

The general syntax for **clvt** commands:

```
clvt action class object name [name=value]...
```

Not all action class combinations require object or name=value pairs.

Not all commands have output. If they do, this is noted.

If the user enters an illegal value, the return code is non-zero. Not all commands produce error messages.

General rules for syntax

[]Material within brackets is optional

{} Material within braces is required

italics with no flag - the object named is required.

| =Indicates an alternative. Use one of the options.

... Indicates that one or more of the kinds of objects or parameters preceding the ellipsis can be entered.

Cluster class operations

Use **clvt** to perform operations on a cluster.

add cluster

Creates a PowerHA SystemMirror cluster with the specified name. If the cluster name already exists, returns non-zero. If you do not supply a name, the cluster will be named cluster_*nodename* where *nodename* is the node where you run the command.

Syntax

```
clvt add cluster [cluster_name]
```

cluster_name The cluster name can contain up to 32 alphanumeric characters and underscores.

Example

Create a cluster named ClusterA:

```
clvt add cluster clusterA
```

delete cluster

Deletes the currently defined cluster. If the specified cluster does not exist or does not match the existing cluster, returns non-zero.

Syntax

```
clvt delete cluster
```

Example

Delete a cluster:

```
clvt delete cluster
```

query cluster

Returns information about the cluster. If the cluster does not exist, returns non-zero.

Syntax

```
clvt query cluster
```

Example

```
clvt query cluster

SECURITY="Standard"
CLUSTER_ID="1146018839"
STATE="ST_INIT"
CLUSTER_NAME="regaa11_cluster"
```

sync cluster

Performs a verification and synchronization of the cluster. Errors found during verification are automatically corrected. If the cluster does not exist, returns non-zero.

Syntax

```
clvt sync cluster
```

Example

To verify and synchronize the cluster:

```
clvt sync cluster
```

discover cluster

Performs cluster discovery of PowerHA SystemMirror related information from nodes configured in the specified cluster. If the cluster does not exist, returns non-zero.

Syntax

```
clvt discover cluster
```

Example

To discover the current cluster:

```
clvt discover cluster
```

Node class operations

Use **clvt** to perform operations on a cluster node.

add node

Adds a node with the specified name to the cluster.

Syntax

```
clvt add node node_name COMMPATH =IPaddress | IPlabel
```

Parameters

Parameter	Description
node_name	Name of a defined cluster node.
PATH	Communication path to the node.

Required objects

The cluster.

Errors

- The cluster is not defined
- The node name already exists
- The maximum number of nodes already exists.

Example

To add a node named node1 to the cluster with its IP address:

```
clvt add node node1 COMMPATH=10.10.2.2
```

delete node

Deletes the specified node from the cluster.

Syntax

```
clvt delete node node_name
```

Parameters

Parameter	Description
node_name	Name of an existing cluster node.

Required objects

- cluster
- node.

Errors

- The cluster is not defined
- The node does not exist.

Example

To delete node2 from the cluster:

```
clvt delete node node2
```

query node

If a node name is specified, the command returns discovered information about the node attributes. If a node name is not specified, the command returns a list of all nodes.

Syntax

```
clvt query node [node_name ]
```

Parameters

Parameter	Description
node_name	Name of an existing cluster node.

Required objects

cluster

Errors

- The cluster is not defined
- The node does not exist.

Output

If node name is supplied, information about the following items are returned:

- Node Name
- Communication Path
- ATTR = (public or private)
- IPADDRESS = boot addresses and interface addresses
- IPLABEL=boot labels and interface labels
- NETMASK=for each path
- NETTYPE=type of networks
- NETWORK=network names.

Examples

```
clvt query node

  maple
  elm
  oak
clvt query node regaa11 | sort
ATTR1="public"
ATTR2="public"
ATTR3="public"
IPADDRESS1="192.168.210.11"
IPADDRESS2="192.168.220.11"
IPADDRESS3="10.70.33.11"
IPLABEL1="regaa11_base10"
IPLABEL2="regaa11_base20"
IPLABEL3="regaa11"
NAME="regaa11"
NETMASK1="255.255.255.0"
NETMASK2="255.255.255.0"
NETMASK3="255.255.0.0"
NETTYPE1="ether"
NETTYPE2="ether"
NETTYPE3="ether"
NETWORK1="net_ether_01"
NETWORK2="net_ether_01"
NETWORK3="net_ether_00"
```

online node

Starts the Cluster Manager on the specified node. If the Cluster Manager is already started the command is ignored.

Syntax

```
clvt online node node_name
```

Parameters

Parameter	Description
node_name	Node to bring online.

Required objects

- cluster
- node.

Errors

- The cluster is not defined
- The node does not exist.

Example

To bring node2 online:

```
clvt online node node2
```

offline node

Stops the Cluster Manager on the specified node. If the Cluster Manager is already stopped the command is ignored.

Syntax

```
clvt offline node node_name
```

Parameters

Parameter	Description
node_name	Name of an existing cluster node.

Required objects

- cluster
- node.

Errors

- The cluster is not defined
- The node does not exist.

Example

To bring node2 offline:

```
Clvt offline node node2
```

Interface class operations

Use **clvt** to perform operations on an interface.

modify interface

Modifies the network to which an interface will be bound.

Syntax

```
clvt modify interface iplabel NETWORK=networkname
```

Parameters

Parameter	Description
iplabel	Name of the interface
networkname	Network to which an interface will be bound.

Required objects

- cluster
- node
- interface.

Errors

- Interface does not exist
- Network does not exist.

Example

To bind the iplabel named mylabel to the network ether2:

```
clvt modify interface mylabel NETWORKNAME=ether2
```

query interface

Returns information about an interface label. If an interface label is not specified, returns a list of all interface labels.

Syntax

```
clvt query interface [interface_label ]
```

Parameters

Parameter	Description
interface_label	The name of the interface

Required objects

cluster

Errors

- The cluster is not defined
- The node does not exist.

Output

If no interface label is supplied, displays a list of interface labels. If an interface label is supplied, returns the following information:

- Interface Label Name
- Interface Type
- Network
- Network Type
- Network Attribute (Public or Private)
- Node
- IP Address
- Hardware Address
- Interface Name.

Examples

To list all interfaces:

```
clvt query interface
  maple_base_10
  maple_base_20
  elm_base_10
  elm_base_20
```

To list the attributes of interface regaa11_base10:

```
clvt query interface regaa11_base10
  INTERFACENAME="en0"
  NETTYPE="ether"
  TYPE="boot"
  ATTR="public"
  IPADDR="192.168.210.11"
  NODE="regaa11"
  GLOBALNAME=""
  HADDR=""
  NETWORK="net_ether_01"
  NAME="regaa11_base10"
```

Network class operations

Use **clvt** to perform operations on a cluster network.

add network

Adds a network to the cluster.

Syntax

```
clvt add network networkname NETWORKTYPE=networktype [NETMASK=netmask]
```

Parameters

Parameter	Description
networkname	Name of the network to add.
NETWORKTYPE	Type of network being added. Choices are ether, XD_ip, or XD_data.
NETMASK	Netmask for the network

Required objects

cluster

Errors

- Cluster does not exist
- Invalid network type
- Invalid netmask.

Examples

Add a network named ether_01 of type ether, with a netmask of 255.255.128:

```
clvt add network ether_01 NETWORKTYPE=ether \
                          NETMASK=255.255.255.128 \
```

delete network

Deletes a previously defined network.

Syntax

```
clvt delete network networkname
```

Parameters

Parameter	Description
networkname	Name of the network to delete.

Required objects

cluster network.

Errors

Network does not exist.

Example

To delete a network named ether1 from the cluster:

```
clvt delete network ether1
```

query network

Returns information about the specified network. If a network name is not specified, returns a list of all networks.

Syntax

```
clvt query network [networkname ]
```

Parameters

Parameter	Description
networkname	Name of the network to delete.

Required objects

cluster

Errors

- The cluster is not defined
- The network does not exist.

Output

If a network name is not supplied, lists all cluster networks. If a network name is supplied, lists the following network attributes:

```
ALIAS="true | false"
ALIAS_HB_ADDR= " "
ALIAS_HB_NETMASK=""
ATTR="public | private"
GLOBALNAME=""
MONITOR_METHOD="default | custom"
NAME="network_name"
NETMASK=""
NETWORK_ID=""
NIMNAME=""
POLLINTERVAL=""
```

Example

To list the cluster networks:

```
clvt query network
  net_XD_data_01
  net_XD_ip_01
  net_ether_01
  token_01
```

To list the attributes of the network named net_ether_01:

```
clvt query network net_ether_01
  ALIAS="aliased"
  ATTR="public"
  POLLINTERVAL="0"
  NIMNAME="ether"
  NETWORK_ID="0"
  GLOBALNAME=""
  NAME="net_ether_01"
  ALIAS_HB_NETMASK=""
  NETMASK="255.255.255.0"
  ALIAS_HB_ADDR=""
  MONITOR_METHOD="default"
```

resource_group class

The resource_group class is a little different in that the **clvt add** action is used to add the resource group to the cluster and the **clvt modify** action is used to either add or modify the attributes and resources of the resource group. Use **clvt** to perform operations on a resource group.

add resource_group

Adds a resource group to a cluster. If the cluster and nodes are not yet defined, adding a resource group defines them and runs cluster discovery. The cluster name is generated from the resource group name. If no startup, fallover, or fallback policies are defined, the defaults are used.

Syntax

```
clvt add resource_group resource_group_name PRIMARYNODES="primary node
list" [STARTUP="startup policy"] [FALLBACK="fall back policy"]
[FALLOVER="fall over policy"]
```

Parameters

Parameter	Description
Resource_group_name	Alphanumeric string of no more than 32 characters.
PRIMARYNODES	Node list for the resource group. The node list is in priority order.
STARTUP	Startup up policy for the resource group. Possible values are: • OHN - Online on home node (Default) • OFAN - Online on first available node • OAAN - Online on all available nodes • OUDP - Online using distribution policy
FALLBACK	Fallback policy for the resource group. Possible values are: • NFB - Never fall back (Default if STARTUP = OAAN) • FBHPN - Fall back to higher priority node in list (Default if STARTUP!= OAAN)
FALLOVER	Fallover policy for the resource group. Possible values are: • FNPN - Fall over to next priority node in the list (Default if STARTUP != OAAN) • FUDNP - Fall over using Dynamic Node Priority • BO - Bring offline (on error node only)(Default if STARTUP = OAAN)

Errors

- Node could not be added to cluster
- Invalid startup policy
- Invalid fallback policy
- Invalid fallover policy.

Example

To add a resource group named myDBapp with a startup policy of OFAN, fallback policy of NFB and fallover policy of FNPN that can run on Nodes Apple and Basket:

```
clvt add resource_group myDBapp NODE_LIST="Apple, Basket"
[STARTUP="OFAN"] [FALLBACK="NFB"] [FALLOVER="FNPN"]
```

delete resource_group

Deletes a specified resource group from the cluster.

Syntax

```
clvt delete resource_group resource_group_name
```

Parameters

Parameter	Description
resource_group_name	Resource group to delete.

Required objects

- cluster
- resource_group.

Errors

- The cluster is not defined
- The resource group does not exist.

Example

To delete resource group MyDBapp:

```
clvt delete resource_group MyDBapp
```

modify resource_group

Adds or modifies the attributes and resources that are part of the resource group.

Syntax

```
clvt modify resource_group <resource group name>
[ SERVICE_LABEL="service1 service2..." ]
[ APPLICATIONS="app1 app2..." ]
[ VOLUME_GROUP="vg1 vg2 ..." ]
[ FORCED_VARYON="true | false" ]
[ VG_AUTO_IMPORT="true | false" ]
[ FILESYSTEM="/fs1 /fs2 ..." ]
[ FS_BEFORE_IPADDR="true | false" ]
[ EXPORT_FILESYSTEM="/expfs1 /expfs2 ..." ]
[ MOUNT_FILESYSTEM="/nfs_fs1 /nfs_fs2 ..." ]
[ NFS_NETWORK="nfs_network" ][ DISK="hdisk1 hdisk2 ..." ]
```

Parameters

Parameter	Description
SERVICE_LABEL	Comma-separated list of defined service labels and addresses
APPLICATIONS	Comma-separated list of defined application controllers
VOLUME_GROUP	Comma-separated list of volume groups to be varied on
FORCED_VARYON	True if force varyon will be used to varyon volume groups
VG_AUTO_IMPORT	= "true | false"
FILESYSTEM	File systems to mount. (Default is ALL)
FS_BEFORE_IPADDR	= "true | false"
EXPORT_FILESYSTEM	The mount points of the file systems and directories that are exported to all nodes in the resource chain when the resource is initially acquired.
NFS_NETWORK	Preferred network for NFS mounts
MOUNT_FILESYSTEM	All nodes in the resource chain that do not currently hold the resource will attempt to NFS-mount these file systems.
DISK	= "hdisk1 hdisk2..."

Required objects

- cluster
- resource_group.

Errors

- The cluster is not defined
- The resource group does not exist
- The service label is not defined
- The application controller does not exist.

Examples

- To modify the volume groups to be varied on in resource group MyDBapp:

```
clvt modify resource_group MyDBGapp VOLUME_GROUP=vgmyDB1,vgmyDB3
```

- To modify the NFS network and application controllers for resource group MyDBapp:

```
clvt modify resource_group myDBapp NFS_NETWORK=ether2
APPLICATIONS=appserv1,appserv3
```

query resource_group

Returns information about a specified resource group. The information returned contains both items from the add action and the modify action. If you do not specify a resource group name, the command returns a list of all resource groups.

Syntax

```
clvt query resource_group [resource_group_name]
```

Parameters

Parameter	Description
resource_group_name	Name of the resource group for which to list information.

Required objects

cluster

Errors

- The cluster is not defined
- The resource group does not exist.

Output

If the name of a non-modified resource group is supplied, output shows the basic information about the resource group:

- Name
- Node list
- Startup, fallover, and fallback policies
- State.

If the name of a resource group that has been modified is supplied, returns complete information about all resource group attributes:

```
Disk
Volume Group
Concurrent Volume Group
```

```
Forced Varyon
Filesystem
Export Filesystem
Shared Tape Resources
Communication Links
Applications
Mount Filesystem
Service Label
VG Auto Import
Mount Filesystem before IP Address
NFS Network
Node Priority Policy
Mount All Fs
Fallback At
Relationship
Secondary Nodes
Primary Nodes
Startup Policy
Fallover Policy
Fallback Policy
Resource Group State
```

Examples

```
clvt query resource_group
  rg1
  rg2
  rg3

clvt query resource_group RG1
  DISK= "hdisk3"
  EXPORT_FILESYSTEM="/FSa1"
  FALLBACK="FBHPN"
  FALLOVER="FNPN"
  FILESYSTEM="/FSa1"
  FORCED_VARYON="false"
  FSCHECK_TOOL="fsck"
  FS_BEFORE_IPADDR="true"
  MOUNT_FILESYSTEM="/mnt1;/FSa1"
  NAME="RG1" NFS_NETWORK=""
  NODES="regaa11 regaa12"
  RECOVERY_METHOD="sequential"
  SERVICE_LABEL="alias_svc1"
  SSA_DISK_FENCING="false"
  STARTUP="OHN" STATE=""
  VG_AUTO_IMPORT="true"
  VOLUME_GROUP="vg1"
```

online resource_group

Brings a specified resource group online. If the resource group is already online, the command is ignored.

Syntax

```
clvt online resource_group resource_group_name
```

Parameters

Parameter	Description
resource_group_name	Name of the resource group to bring online.

Required objects

- cluster
- resource_group.

Errors

- The cluster is not defined
- The resource group does not exist.

Example

To bring the resource group named MyDBapp1 online:

```
clvt online resource group MyDBapp1
```

offline resource_group

Brings a resource group offline. If the resource group is already offline, the command is ignored. The resource group name must be specified.

Syntax

```
clvt offline resource_group resource_group_name
```

Parameters

Parameter	Description
resource_group_name	Name of the resource group to bring online.

Required objects

- cluster
- resource_group.

Errors

- The cluster is not defined
- The resource group does not exist

Example

To bring the resource group named MyDBapp1 offline:

```
clvt offline resource group MyDBapp1
```

service_ip class

Use **clvt** to do perform operations on a service IP

add service_ip

Adds a shared or node bound service IP label to the cluster. This label can later be added to a resource group.

Syntax

```
clvt add service_ip ip_label_name NETWORKNAME=networkname SHARED="true |
false" [NODENAME=nodename]
```

Parameters

Parameter	Description
ip_label_name	Name of the IP label.
NETWORKNAME	Name of the network on which the IP label and address will be attached.
SHARED	"True" if the service IP label is configurable on multiple nodes. "False "if the service IP label is node bound.
NODENAME	Node where the service IP label will be bound.

Required objects

- cluster
- network.

Errors

- Cluster does not exist
- Invalid network name
- Undefined node name
- Unshared service IP label requires node name.

Example

```
clvt add service_ip {LabelA} NETWORK={ServiceNetwork} \
                             SHARED=false \
                             NODENAME={NodeA}
```

delete service_ip

Deletes a previously defined service IP label.

Syntax

```
clvt delete service_ip service_IP_label
```

Parameters

Parameter	Description
service_IP_label	Service_IP_label to delete.

Required objects

- cluster
- service_ip.

Example

Delete a service IP label named LabelA:

```
Clvt delete service_ip LabelA
```

query service_ip

Returns information about the specified service IP label. If a service IP label is not specified, returns a list of all service IP labels.

Syntax

```
clvt query service_ip [service_IP_label]
```

Parameters

`service_IP_label`

Required objects

cluster

Output

If a service IP label is supplied, the following information is returned:

- Service IP Label Name
- IP Label Type
- Network
- Network Type
- Network Attribute
- Node
- IP address
- Hardware Address
- Interface Name
- Global Name.

Examples

```
clvt query service_ip
  alias_svc1
  alias_svc2
  maple_base10
  elm_base10
  oak_base10

clvt query service_ip alias_svc1
  INTERFACENAME=""
  NETTYPE="ether"
  TYPE="service"
  ATTR="public"
  IPADDR="192.168.250.1"
  NODE=""
  GLOBALNAME="ignore"
  HADDR=""
  NETWORK="net_ether_01"
  NAME="alias_svc1"
```

Application controller class use

clvt to perform operations for an application controller.

add application

Adds an application controller to the cluster.

This can later be added to a resource group.

Syntax

clvt add application *application_controller_name* STARTPATH=start_script_path STOPPATH=stop_script_path

Parameters

Parameter	Description
Application_controller_name	Alphanumeric string of no more than 32 characters.
STARTPATH	Full path for the start script for the application controller.
STOPPATH	Full path for the stop script for the application controller.

Required Objects

cluster

Errors

- Cluster does not exist
- Application monitor is not defined.

Example

To add an application controller named App1: clvt add application App1 STARTSCRIPT=/full pathnameX \STOPSCRIPT=/full pathnameY

delete application

Deletes a previously defined application controller.

Syntax

```
clvt delete application application_controller_name
```

Parameters

Parameter	Description
application_controller_name	Application controller to delete.

Required objects

- cluster
- application.

Errors

The application controller does not exist.

Example

To delete the application controller named MyApp:

```
clvt delete application MYApp
```

query application

Returns information about the specified application controller. If no application controller is specified, returns a list of all application controllers.

Syntax

```
clvt query application [application_controller_name]
```

Parameters

```
application_controller_name
```

Required objects

cluster

Errors

- The cluster is not defined
- The application controller does not exist.

Output

If no application controller is supplied, a list of application controllers. If an application controller is supplied, the following information is returned:

```
Application Controller Name
Start Script
Stop Script
Application Monitors (if configured)
```

Examples

```
clvt query application
  app_srv_1
  db2
  websphere

clvt query application App1
  STARTSCRIPT="/xxxx"
  ASSOCIATEDMONITORS="AppMon"
  NAME="App1"
  STOPSCRIPT="/yyyy"
```

application_monitor class

Use **clvt** to perform operations for an application monitor.

add application_monitor

Adds an application monitor to the cluster. This can later be added to an application controller.

Syntax

```
clvt add application_monitor <application monitor name>
         APPLICATIONMONITORTYPE="Process | Custom"
         APPLICATIONSERVERNAME="appserver1"
         MONITORMODE="longrunning | startup | longrunningandstartup"
         FAILUREACTION="notify | fallover"
         STABILIZATION="1 .. 3600"
         RESTARTCOUNT="0 .. 100"
        [ PROCESSMONITORS="pmon1 dbmon ..." ]
        [ INSTANCECOUNT="1 .. 1024" ]
        [ MONITORMETHOD="/script/to/monitor" ]
        [ MONITORINTERVAL="1 .. 1024" ]
        [ HUNGSIGNAL="1 .. 63" ]
        [ RESTARTINTERVAL="1 .. 3600" ]
```

Parameters

Parameter	Description
APPLICATIONMONITORTYPE	Monitor type. Options are "Process" and "Custom"
MONITORMODE	Monitor mode. Options are "longrunning", "startup", or "both"
APPLICATIONSERVERNAME	Application controller name being monitored
PROCESSMONITORS	Process names to be monitored
INSTANCECOUNT	Count of processes to be monitored
MONITORMETHOD	Custom application monitor script
MONITORINTERVAL	The monitor method will be run periodically at this interval (in seconds).
HUNGSIGNAL	The signal sent to stop the Monitor Method if it doesn't return within Monitor Interval seconds. The default is SIGKILL(9).
STABILIZATION	Time to wait for monitor to stabilize
RESTARTCOUNT	Number of times to restart the application controller
RESTARTINTERVAL	Time that the application must remain stable before resetting RESTARTCOUNT
FAILUREACTION	Action to take on failure detection. Options are "notify" or "fallover"

Required objects

cluster

Errors

- The cluster is not defined
- Application controller is not defined.

Examples

To add a process monitor named pmon1:

```
clvt add application monitor pmon1
APPLICATIONSERVERNAME="appserver1"
 MONITORMODE="longrunning"
  FAILUREACTION="notify"
  PROCESSMONITORS="pmon1."
PROCESSOWNER="<username>"
 INSTANCECOUNT="5"
  STABILIZATION="50"
  RESTARTCOUNT="5"
RESTARTINTERVAL="50"
NOTIFYMETHOD="</script/name>"
```

To add a custom monitor named cmon1:

```
clvt add application monitor cmon1
APPLICATIONSERVERNAME="appserver1"
  MONITORMODE="longrunningandstartup"
  MONITORMETHOD="/script/to/monitor"
  MONITORINTERVAL="1024"
  HUNGSIGNAL="60"
  STABILIZATION="3600"
  RESTARTCOUNT="100"
  RESTARTINTERVAL="2400"
  FAILUREACTION="fallover"
```

delete application_monitor

Deletes a previously defined application monitor.

Syntax

```
clvt delete application_monitor application_monitor_name
```

Parameters

Parameter	Description
application_monitor_name	The application monitor to delete.

Required objects

- cluster
- application_monitor.

Errors

The application monitor does not exist.

Example

To delete an application monitor named MyMonitor:

```
clvt delete application monitor MyMonitor
```

query application_monitor

Returns information about the specified application monitor. If an application monitor is not specified, returns a list of all application monitors.

Syntax

```
clvt query application_monitor [application_monitor_name]
```

Parameters

Parameter	Description
app_monitor_name	The application monitor to delete.

Required objects

- cluster

Errors

- The application monitor does not exist
- The cluster is not defined.

Examples

```
clvt query application_monitor
app_srv_1_mon
db2_mon
websphere_mon

clvt query application_monitor App1Mon1
MONITORINTERVAL="30"
CLEANUPMETHOD="/xxxx"
RESTARTCOUNT="3"
HUNGSIGNAL="9"
MONITORMODE="longrunning"
APPLICATIONSERVERNAME="App1"
RESTARTMETHOD="/yyyy"
FAILUREACTION="notify"
MONITORMETHOD="/xxxx"
```

```
APPLICATIONMONITORTYPE="user"
NAME="App1Mon1"
RESTARTINTERVAL="132"
STABILIZATION="10"  s
```

Resource group temporal_dependency class

Use **clvt** to modify or query a resource group temporal dependency configuration.

modify temporal_dependency

Modifies a parent and child resource group dependency.

Syntax

```
clvt modify temporal_dependency parent_resource_group
[RESOURCEGROUPCHILD=child_resource_groups]
```

Parameters

Parameter	Description
parent_resource_group	Parent resource group.
RESOURCEGROUPCHILDREN	Space-delimited list of child resource groups. If none are specified, then all child resource group associations for the specified parent resource group are removed. Specifying a new list replaces the current one.

Required objects

- cluster
- resource group.

Errors

- The cluster is not defined
- The parent resource group does not exist
- The child resource group does not exist.

Example

To add a child resource group RG2c2 to RG2, thus modifying a temporal dependency that had only one child (RG2c1) previously:

```
clvt modify temporal_dependency RG2 RESOURCEGROUPCHILDREN="RG2c1 RG2c2"
```

query temporal dependency

Lists the parent and child dependencies for the specified resource group.

Syntax

```
clvt query temporal_dependency resource group MODE="children | parents"
```

Parameters

Parameter	Description
resource group	Resource group to display
MODE	How output is displayed: by "parents" or by "children."

Required objects

- cluster
- resource group.

Output

A list of parent and child dependencies.

Examples

```
clvt query temporal_ dependency RG2c1 MODE=parents
  RG2

clvt query temporal_dependency RG2 MODE=children
  RG2c1
  RG2c2
```

Resource group location_dependency class

Use **clvt** to modify or query a resource group location dependency configuration.

modify location_dependency

Modify a location dependency between resource groups. The only location dependency type supported is SAME_NODE.

Syntax

```
clvt modify location_dependency SAME_NODE
RESOURCEGROUPLIST=resource_group_list
```

Parameters

Parameter	Description
SAME_NODE	Location dependency type.
RESOURCEGROUPLIST	Space-delimited list of resource groups

Required objects

- cluster
- resource group.

Errors

- The cluster is not defined
- The resource group does not exist.

Examples

To set up a location dependency so that RG1 and RG2 stay on the same node:

```
clvt modify location_dependency SAME_NODE\ RESOURCEGROUPLIST="RG1 RG2"
```

query location_dependency

Returns a list of the sets of resource groups that are configured to always be on the same node.

Syntax

```
clvt query location_dependency [SAME_NODE]
```

Parameters

Parameter	Description
SAME_NODE	Type of location dependency to display.

Required objects

cluster

Examples

Where was_rg_1 and ihs_rg_1 are defined to stay on the same node, and was_rg_2 and ihs_rg_2 are defined to stay on the same node:

```
clvt query location dependency SAME_NODE
  was_rg_1 ihs_rg_1
  was_rg_2 ihs_rg_2
```

file_collection class

Use **clvt** to do these operations on a file collection.

add file_collection

Adds a file collection to the cluster.

Syntax

```
clvt add file_collection file_collection_name
[ISPROPOGATEFILESDURINGSYNC="true | false"]
[ISPROPOGATEAUTOWHENDETECTED="true | fslse"] [FILES= "/path/to/file1
/path/to/file2..."]
```

Parameters

Parameter	Description
file_collection_name	Alphanumeric string of not more than 32 characters.
ISPROPOGATEFILESDURINGSYNC	"true" or "false". Cause files to propagate from the cluster when and where synchronization occurs. Default is false.
ISPROPOGATEAUTOWHENDETECTED	"true" or "false". Cause files to propagate when changes occur. Default is false.
FILES	Space-delimited list of full pathname of files to be managed by the specified file collection.

Required objects

cluster

Errors

The cluster is not defined.

Example

To add a file collection named FC1 With files xxxx and yyyy that are propagated automatically when detected during a sync:

```
clvt add file_collection FC1 FILES="/xxxx /yyyy"\
ISPROPOGATEDFILEDURINGSYNC=true \
ISPROPOGATEAUTOWHENDETECTED=true
```

delete file_collection

Deletes a specified file collection.

Syntax

```
clvt delete file_collection file_collection_name
```

Parameters

Parameter	Description
file_collection_name	File collection to delete.

Required objects

- cluster
- file_collection.

Errors

- The cluster is not defined
- The file collection does not exist.

Example

Delete a file collection named FC1

```
clvt delete file collection FC1
```

query file_collection

Returns information about the specified file collection. If a file collection is not specified, returns a list of all file collections.

Syntax

```
clvt query file_collection [file_collection_name]
```

Parameters

Parameter	Description
file_collection_name	Name of file collection to query.

Required objects

cluster

Errors

- The cluster is not defined
- The file collection does not exist.

Output

If no file collection is supplied, the output is a list of file collections. If a file collection is supplied, the following information is returned: File Collection Name File Collection Description Propagate during synced Propagate when changed Files - Each file is separated by a space

Example

```
clvt query file_collection FC1
  ISPROPOGATEDFILEDURINGSYNC="false"
  FILES="/xxxx /yyyy"
  ISPROPOGATEAUTOWHENDETECTED="false"
  DESCRIPTION="File_Collection_FC1"
  NAME="FC1"
```

Sample Smart Assist program

These topics provides a sample Smart Assist program, based on the General Application Smart Assist (GASA).

Overview

The General Application Smart Assist (GASA) is a pre-installed Smart Assist that comes with PowerHA SystemMirror. Its intended purpose is for configuring applications that do not already have a target Smart Assists, but can be easily managed via start and stop scripts.

The general Smart Assist requests the minimal amount of information from the user and will configure the PowerHA SystemMirror resource group and application controller necessary to support the target application. The GASA Smart Assist has only one Smart Assist component; more complicated Smart Assists will have several components. All of the concepts and tasks associated with creating a single component Smart Assist can be extended and applied to constructing a Smart Assist with multiple components.

Functionality of the sample program

The add screen for the Smart Assist associates a resource group to a set of participating nodes selected by the user. Inside the resource group is an application controller constructed via the Smart Assist In the sections that follow we'll dissect the General Application Smart Assist into the sub-tasks required to create a new Smart Assist.

Installing the sample program

The General Application Smart Assist is installed via the **cluster.es.assist.common** fileset, several files are copied to the local filesystem.

These files are:

```
/usr/es/sbin/cluster/sa
                  /gasa
                        ./sbin
                              add
                              discovery
                              modify
                              smit.utils
```

Script Name	Description
add	Adds new instances of the GASA application to the PowerHA SystemMirror cluster configuration
discovery	Runs discovery on the local cluster node, determines if the Smart Assist component can be configured. Script typically asks the question: Is the application installed on the local node?
modify	Modifies the existing PowerHA SystemMirror application instance
smit_util	SMIT utilities for performing cmd_to_list, cmd_to_discover, cmd_to_classify operations

Each of the above scripts is written in ksh93; all of the source code to the GASA Smart Assist is accessible. It is recommended to package all of the scripts under the same directory hierarchy:

```
/usr/es/sbin/cluster/sa/<Smart Assist name>/sbin/
```

There are no functional limitations imposed if this hierarchy is not followed; it is simply the recommended practice.

File permissions

Because PowerHA SystemMirror requires the root user to configure PowerHA SystemMirror, all of the Smart Assist scripts and binaries require root permissions to execute, and typically will have the permissions set to 500, owner root, group system. This requirement is especially true of the **clvt** API commands. All of the **clvt** API commands manipulate the ODMs and therefore require root privileges.

Smart Assist registration

After the files are installed on the local node, the Smart Assist installation process should call the **claddsa** script to register the various attributes of the Smart Assist components with the application framework. With GASA this is accomplished with the following command:

```
/usr/es/sbin/cluster/sa/sbin/claddsa -a zz_Other -c GASA \
SMARTASSIST_VERSION="1.0" \
  COMPONENT_ID="GASA" \
  SMARTASSIST_ID="zzOther" \
  DISCOVERY_COMMAND="/usr/es/sbin/cluster/sa/gasa/sbin/discovery" \
  SMIT_ADD="clsa_gasa_add" \
  SMIT_ADD_TYPE="d" \
  SMIT_MODIFY="clsa_gasa_modify" \
  SMIT_MODIFY_TYPE="d"
```

Note that for GASA there is only one Smart Assist component. If additional components are required, multiple calls to the above command would be necessary for each component.

The result of calling the **claddsa** command is a set of HACMPsa ODM entries. These entries can be seen by using the command:

```
odmget -q "sa_id=zzOther and component_id=GASA" HACMPsa
```

On any PowerHA SystemMirror cluster node where the **cluster.es.assist.common** fileset has been installed, the above ODM entries will be installed.

Deinstalling the sample program

The Smart Assist component entries added via the **claddsa** command need to be removed during deinstallation. Call the **clrmsa** command to remove the entries.

The GASA Smart Assist calls the following command to remove its entries from HACMPsa:

```
/usr/es/sbin/cluster/sa/sbin/clrmsa -a zzOther
```

Unlike the **claddsa** command, the **clrmsa** command can remove all entries for the entire Smart Assist (as shown above), or for a particular component of the Smart Assist by specifying the **-c** <*component id* > flag.

Once the entries are removed, any existing application instance constructed with the uninstalled Smart Assist will have its references to the application removed. Users will be unable to modify the application from within the Smart Assist UI, but will still be able to modify or delete the constructed resource groups and associated resources.

Command to discover Smart Assist components

Recall that the DISCOVERY_COMMAND is per Smart Assist per component. For the GASA Smart Assist there is only one component, and therefore only one command to execute.

The command is executed when the user selects the SMIT menu option:

Add an Application to the PowerHA SystemMirror Configuration

Once the user selects that SMIT menu, the framework runs the DISCOVERY_COMMAND of every Smart Assist component on every available node in the cluster where that discovery command exists. The command to discover determines whether the target application is installed and suitable for use with the Smart Assist. The GASA Smart Assist is very generic; it allows all PowerHA SystemMirror cluster nodes in the Smart Assist and therefore performs no validation.

The output of the discovery command is one line that contains information in the following form:

```
Smart Assist Name in I8N format :
Smart Assist Identifier :
Component Name in I8N form :
Component Identifier :
 0 | 1
```

The trailing 0, or 1 value determines whether the Smart Assist can be used on the local node. If the discovery command prints 0, then the Smart Assist component cannot be used on the local node; a value of 1 means the Smart Assist component can be used on the local node. For the GASA Smart Assist a line is printed as shown below:

```
Other Applications:zzOther:General Application Smart Assist:GASA:1
```

The first and third column values are internationalized using dspmsg and a message catalog.

Add application instance functionality

There are two pieces to constructing a Smart Assist that allow users to add new instances of an application to the PowerHA SystemMirror configuration, the SMIT component, which include the dialog and smit fields, and the script that performs the add operation.

Both of these operate on a single Smart Assist component and are described in the sections that follow.

Creating the add SMIT screen

The add SMIT screens identifiers for the GASA Smart Assist are defined in the call to **claddsa**:

```
SMIT_ADD="clsa_gasa_add"
```

and

```
SMIT_ADD_TYPE="d"
```

These entries point to a SMIT sm_cmd_hdr element, whose children are sm_cmd_opt entries that form the SMIT dialog fields. These fields are where the user enters information related to the application, such as the name, service IP label to use, and the start and stop scripts that are required by the Smart Assistant. The entries used to construct the dialog are shown in the section SMIT general application Smart Assist add stanzas.

The application framework will call the following SMIT command header entry "clsa_gasa_add", which provides the General Application Smart Assist SMIT screen:

General Application Smart Assist - Add Screen

```
              General Application Smart Assist

 Type or select values in entry fields.


 Press Enter AFTER making all desired changes.

                                    [Entry Fields]
* Application Name                  []
* Primary Node                      [node name displays here] +
* Takeover Nodes                    [node names display here] +
* Application Controller Start Script  []
* Application Controller Stop Script   []
* Service IP Label                  [] +
```

Not all Smart Assists transition to a sm_cmd_hdr; in some cases a sm_name_hdr is required to provide a select list prior to entering the dialog. In those cases the SMIT_ADD_TYPE would equal 'n' for name selector, and the SMIT_ADD field for the component would point to the SMIT id of the sm_name_hdr.

Creating the add script

Each of the field entries that are provided above correspond to an argument that will be passed to the add script: GASA Smart Assist add script: /usr/es/sbin/cluster/sa/gasa/add

Field Name	Argument	Discovery Field Name (disc_name)
Application Name	-a name	N/A
Primary Node	-P node	Primary
Takeover Nodes	-T nodes	takeover
Start Script	-R script	N/A
Stop Script	-O script	N/A
Service IP label	-S ip_label	N/A
	-C component_id	N/A
	-s smartassist_id	N/A

GASA Add Screen Field Names to Arguments

The GASA add script can be re-used in your Smart Assist. Note that the caller provides the Smart Assist identifier and the component identifier. By changing the Smart Assist identifier -s, and the component identifier -C, you can use the existing GASA add script as the basis for your Smart Assist components add script. The add script performs the following sequence of operations:

1. Checks the user-entered application name for a duplicate entry using the **claddsaapp** API command.
2. Determines if the Application Controller already exists using:
   ```
   clvt query application
   ```
3. Determines if the resource group already exists using:
   ```
   clvt query resource_group
   ```
4. Calls **clsapre**
5. Adds the application controller to the PowerHA SystemMirror configuration.
   ```
   clvt add application $SERVERNAME \
      STARTSCRIPT=$STARTSCRIPT \
      STOPSCRIPT=$STOPSCRIPT
   ```
6. Adds the resource group to the PowerHA SystemMirror configuration.

```
clvt add resource_group "$RG_NAME" \
  PRIMARYNODES="$PRIMARY $TAKEOVER" \
  STARTUP="OHN" \
  FALLBACK="NFB" \
  FALLOVER="FNPN"
```

7. Creates the service IP label if it is not already defined.

   ```
   clvt add service_ip $SERVICE_LABEL
   ```

8. Associates the service IP label and any available (shared) volume groups amongst the participating nodes (primary and takeover).

   ```
   clvt modify resource_group "$RG_NAME" \
     APPLICATIONS="$NAME" \
     SERVICE_LABEL="$IP" \
     VOLUME_GROUP="$VGS" \
     VG_AUTO_IMPORT="true" \
     FORCED_VARYON="false" \
     FILESYSTEM=""
   ```

9. Registers the application name resource group with the Smart Assist identifier and component identifier. This ensures that when the user goes to select the particular application instance to modify, the framework knows what type of application it is associated with.

   ```
   claddsaapp -a $APPLICATION_NAME \
   APPLICATION_NAME="$APPLICATION_NAME"
   RESOURCE_GROUP="$RG_NAME" \
   SMARTASSIST_ID="$SMARTASSIST_ID" \
   COMPONENT_ID="$COMPONENT_ID"
   ```

   The **claddsaapp** command could contain other information unrelated to the application framework, variables, and values specific to the application instance such as a path to the application, etc. Name value pairs can overlap, the same name can be used more than once.

10. Calls **clsapost -v** which will run cluster verification

After all of the above steps are completed, the application instance is associated (registered) to the Smart Assist component and the user can then perform modify or delete operations on the application and all of its associated resources as a whole.

Registration script

The zzOther (GASA) Smart Assist does not make use of the REGISTRATION_COMMAND for the GASA component. If it did, when the user added the Smart Assist and the add script called **claddsaapp** to associate the application instance properties with the application name, the registration command would be invoked as specified in the HACMPsa ODM stanza.

Related concepts:

"SMIT general application Smart Assist add stanzas" on page 55
This section lists the commented SMIT add stanzas for the GASA Smart Assist.

Modify application instance functionality

The modify functionality is similar to the add functionality. The user selects a particular application instance to modify, based on the application instance Smart Assist identifier and component identifier.

Creating the modify SMIT screen

The modify SMIT screens identifiers for the GASA Smart Assist are defined in the call to **claddsa**

```
SMIT_MODIFY="clsa_gasa_modify"
```

and

```
SMIT_MODIFY_TYPE="d"
```

Identical in approach to the add screen, the application framework will call the clsa_gasa_modify dialog, or name selector based on the entry in the HACMPsa ODM.

What is different about the modify screen is that the framework will also pass along the application_id as a cooked field name, along with the Smart Assist identifier sa_id and the component_id of the application to the dialog or name selector.

The application framework will call the following SMIT command header entry "clsa_gasa_modify", which provides the SMIT screen **Make Applications Highly Available (Use Smart Assists)**:

General Application Smart Assist - Selector Screen

```
          Make Applications Highly Available (Use Smart Assists)

Move cursor to desired item and press Enter.

 Add an Application to the PowerHA SystemMirror Configuration
 Change/Show an Application's PowerHA SystemMirror Configuration
 Remove an Application from the PowerHA SystemMirror Configuration
 Manage Your Applications
 Change/Show the Resources Associated with Your Application
 Test Your Application for Availability

+"""""""""""""""""""""""""""""""""""-+
| Select an Application from the List of Discovered Applications Below
|
| Move cursor to desired item and press Enter.
|
| example_app
+"""""""""""""""""""""""""""""""""""""""""""""""""-+
```

The following cooked field names are passed along to the dialog:

Item	Description
application_id	=example_app
sa_id	=zzOther
component_id	=GASA

The user's SMIT session will transition onto the clsa_gasa_modify dialog:

General Application Smart Assist Modify Dialog

```
     General Application Smart Assist
Type or select values in entry fields.

Press Enter AFTER making all desired changes.

[Entry Fields]
* Application Name             example_app

* Primary Node                 [nodeA] +

* Takeover Nodes               [nodeB nodeC] +

* Application Controller Start     [/example_app/start]
Script

* Application Controller Stop      [/example_app/stop]
Script

* Service IP Label             [service1] +
```

Typically the modify screen of the Smart Assist will look identical to the add screen, what differs is that some of the fields are not editable. Note that in the above figure the application name cannot be changed. The user however can change all other aspects of the application's configuration within SMIT. See SMIT general application Smart Assist modify stanzas for The ODM stanzas.

Note that the fields are already populated when the user enters the modify screen. This is accomplished via the cmd_to_discover method in the sm_cmd_hdr ODM stanza. In the case of the GASA Smart Assist the smit_util script gathers the participating nodes from the associated resource group, and extracts the service IP label, and application start / stop scripts by using the clvt API. A similar approach would be taken for all Smart Assists.

Creating the modify script

In the case of the GASA Smart Assist all of the requisite resources constructed in the add phase are removed, and the add script is called again. In more complicated Smart Assists the modify script would only change those properties of the application that were modified by the user. The GASA first calls the **clsapre** script, then uses the **clvt** API to remove PowerHA SystemMirror components from the cluster configuration. The add script is then called and the application resources are added to the PowerHA SystemMirror configuration. Again, at the tail end of the add script the command **clsapost -v** is called to run verification. If the modify script were to only modify the resources, it should call **clsapost -v** directly.

Related concepts:

"SMIT general application Smart Assist modify stanzas" on page 59
This section lists the commented SMIT Modify stanzas for the GASA Smart Assist.

Deleting application instances

All of the operations required to remove an application instance from the cluster configuration are self contained within the Smart Assist framework. No additional code is required to simply remove the instance.

Deregister script

The GASA does use the DEREGISTRATION_COMMAND, however if one were required the developer only need add the entry to **claddsa** call. Once the user selected an application instance to remove from the cluster configuration the DEREGISTRATION_COMMAND script would be executed prior to actually removing the PowerHA SystemMirror resources. This allows the Smart Assist developer to restore application configuration settings, or provide other application restoration prior to removing PowerHA SystemMirror control of the application.

SMIT general application Smart Assist add stanzas

This section lists the commented SMIT add stanzas for the GASA Smart Assist.

```
# Cooked Field: nodes
#
# Note that the text "nodes" in the cmd_to_discover_postfix is part
# of #the cooked fields from the Smart Assist screens that precede this
# dialog. Users will #select a Smart Assist to instantiate, once
# selected a list of nodes where that smart #assist can be
# used will be provided to the SMIT screens that follow. The "nodes"
# cooked field contains a comma ',' delimited list of node names
sm_cmd_hdr:
  id = "clsa_gasa_add"
  option_id = "clsa_gasa_add_opts"
  has_name_select = ""
  name = "General Application Smart Assist"
  name_msg_file = "cluster.cat"
  name_msg_set = 51
  name_msg_id = 17
  cmd_to_exec = "/usr/es/sbin/cluster/sa/gasa/sbin/add"
```

```
  ask = ""
  exec_mode = ""
  ghost = ""
  cmd_to_discover = "/usr/es/sbin/cluster/sa/gasa/sbin/smit_util
add_discover"
  cmd_to_discover_postfix = "nodes"
  name_size = 0
  value_size = 0
  help_msg_id = "42,10"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Application Name
#
# This is the name that PowerHA SystemMirror will use to uniquely identify the
# collection of PowerHA SystemMirror resource groups, application controller, and other
# related resources
# Users can modify / delete an existing application given its name
# All Smart Assists must contain an application name
#
sm_cmd_opt:
  id_seq_num = "10"
  id = "clsa_gasa_add_opts"
  disc_field_name = ""
  name = "Application Name"
  name_msg_file = "cluster.cat"
  name_msg_set = 43
  name_msg_id = 32
  op_type = ""
  entry_type = "t"
  entry_size = 24
  required = "+"
  prefix = "-a"
  cmd_to_list_mode = ""
  cmd_to_list = ""
  cmd_to_list_postfix = ""
  multi_select = ""
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "30,3"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Primary Node:
#
# For the GASA Smart Assist there exists a separation of primary and
# takeover nodes.
#
# The primary node is where the application will be brought online
# initially and will be the first node in the participating nodes of the
# constructed PowerHA SystemMirror resource group.
#
sm_cmd_opt:
  id_seq_num = "20"
  id = "clsa_gasa_add_opts"
  disc_field_name = "primary"
  name = "Primary Node"
  name_msg_file = "cluster.cat"
  name_msg_set = 51
  name_msg_id = 15
  op_type = "l"
```

```
  entry_type = "t"
  entry_size = 33
  required = "+"
  prefix = "-P"
  cmd_to_list_mode = ""
  cmd_to_list = "/usr/es/sbin/cluster/sa/gasa/sbin/smit_util list"
  cmd_to_list_postfix = "nodes"
  multi_select = ""
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "42,8"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Takeover Nodes:
#
# List of nodes that participate in the constructed resource group
#
# Note that the SMIT cmd_to_list command uses our internal smit_util
# command, which internally calls "clvt query node"
#
sm_cmd_opt:
  id_seq_num = "30"
  id = "clsa_gasa_add_opts"
  disc_field_name = "takeover"
  name = "Takeover Nodes"
  name_msg_file = "cluster.cat"
  name_msg_set = 51
  name_msg_id = 16
  op_type = "l"
  entry_type = "t"
  entry_size = 0
  required = "+"
  prefix = "-T"
  cmd_to_list_mode = ""
  cmd_to_list = "/usr/es/sbin/cluster/sa/gasa/sbin/smit_util list"
  cmd_to_list_postfix = "nodes"
  multi_select = ","
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "42,9"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Application Controller Start Script
#
# Script name used to start the application
#
# No default value
# This property is required
#
sm_cmd_opt:
  id_seq_num = "40"
  id = "clsa_gasa_add_opts"
  disc_field_name = ""
  name = "Application Controller Start Script"
  name_msg_file = "cluster.cat"
```

```
  name_msg_set = 43
  name_msg_id = 4
  op_type = ""
  entry_type = "t"
  entry_size = 256
  required = "+"
  prefix = "-R"
  cmd_to_list_mode = ""
  cmd_to_list = ""
  cmd_to_list_postfix = ""
  multi_select = ""
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "30,4"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Application Controller Stop Script
#
# Script name used to start the application
#
# No default value
# This property is required
#
sm_cmd_opt:
  id_seq_num = "50"
  id = "clsa_gasa_add_opts"
  disc_field_name = ""
  name = "Application Controller Stop Script"
  name_msg_file = "cluster.cat"
  name_msg_set = 43
  name_msg_id = 5
  op_type = ""
  entry_type = "t"
  entry_size = 256
  required = "+"
  prefix = "-O"
  cmd_to_list_mode = ""
  cmd_to_list = ""
  cmd_to_list_postfix = ""
  multi_select = ""
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "30,5"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Service IP label
#
# The user must choose a service IP label to participate in the
# constructed HACM resource group
#
# The command to list cllsserviceips can be used by other Smart Assists
# to provide a list of pre-configured service IP labels, and available
# labels that can be defined to PowerHA SystemMirror.
#
sm_cmd_opt:
```

```
  id_seq_num = "60"
  id = "clsa_gasa_add_opts"
  disc_field_name = ""
  name = "Service IP Label"
  name_msg_file = "cluster.cat"
  name_msg_set = 43
  name_msg_id = 6
  op_type = "l"
  entry_type = "t"
  entry_size = 32
  required = "+"
  prefix = "-I"
  cmd_to_list_mode = "1"
  cmd_to_list = "/usr/es/sbin/cluster/sa/sbin/cllsserviceips"
  cmd_to_list_postfix = ""
  multi_select = "n"
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "30,6"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""
```

SMIT general application Smart Assist modify stanzas

This section lists the commented SMIT Modify stanzas for the GASA Smart Assist.

```
#
# Modify SMIT Dialog
#
# The smit_util modify_discover uses the application_id to determine
# what PowerHA SystemMirror components are associated with the application instance and
# produces cmd_to_discover output of the form:
#
# /usr/es/sbin/cluster/sa/gasa/sbin/smit_util modify_discover exampleapp
#
# #application:primary:takeover:start:stop:ip
# exampleapp:nodeA:nodeB
# nodeC:/exampleapp/start:/exampleapp/stop:service1
#
sm_cmd_hdr:
  id = "clsa_gasa_modify"
  option_id = "clsa_gasa_modify_opts"
  has_name_select = ""
  name = "General Application Smart Assist"
  name_msg_file = "cluster.cat"
  name_msg_set = 51
  name_msg_id = 17
  cmd_to_exec = "/usr/es/sbin/cluster/sa/gasa/sbin/modify"
  ask = ""
  exec_mode = ""
  ghost = ""
  cmd_to_discover = "/usr/es/sbin/cluster/sa/gasa/sbin/smit_util
modify_discover"
  cmd_to_discover_postfix = "application_id"
  name_size = 0
  value_size = 0
  help_msg_id = "42,11"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Application Name
#
```

```
# This is the name that PowerHA SystemMirror will use to uniquely identify the
#collection of PowerHA SystemMirror resource groups, application controllers, and other
#related resources.
# Users can modify / delete an existing application given its name
# All Smart Assists must contain an application name
#
# The name cannot be modified in the modify dialog.
#
sm_cmd_opt:
  id_seq_num = "10"
  id = "clsa_gasa_modify_opts"
  disc_field_name = "application"
  name = "Application Name"
  name_msg_file = "cluster.cat"
  name_msg_set = 43
  name_msg_id = 32
  op_type = ""
  entry_type = "n"
  entry_size = 24
  required = "+"
  prefix = "-a"
  cmd_to_list_mode = ""
  cmd_to_list = ""
  cmd_to_list_postfix = ""
  multi_select = ""
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "30,3"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Primary Node:
#
# For the GASA Smart Assist there exists a separation of primary and
# takeover nodes.
#
# The primary node is where the application will be brought online
# initially and will be the first node in the participating nodes of the
# constructed PowerHA SystemMirror resource group.
#
sm_cmd_opt:
  id_seq_num = "20"
  id = "clsa_gasa_modify_opts"
  disc_field_name = "primary"
  name = "Primary Node"
  name_msg_file = "cluster.cat"
  name_msg_set = 51
  name_msg_id = 15
  op_type = "l"
  entry_type = "t"
  entry_size = 33
  required = "+"
  prefix = "-P"
  cmd_to_list_mode = ""
  cmd_to_list = "cmd_to_list() {\n\
           /usr/es/sbin/cluster/utilities/clvt query node\n\
           return 0\n\
  }\n\
  cmd_to_list"
  cmd_to_list_postfix = ""
  multi_select = ""
  value_index = 0
```

```
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "42,8"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Takeover Nodes:
#
# List of nodes that participate in the constructed resource group
#
# Note that the SMIT cmd_to_list command uses our internal smit_util
# command, which internally calls "clvt query node"
#
  sm_cmd_opt:
  id_seq_num = "30"
  id = "clsa_gasa_modify_opts"
  disc_field_name = "takeover"
  name = "Takeover Nodes"
  name_msg_file = "cluster.cat"
  name_msg_set = 51
  name_msg_id = 16
  op_type = "l"
  entry_type = "t"
  entry_size = 0
  required = "+"
  prefix = "-T"
  cmd_to_list_mode = ""
  cmd_to_list = "cmd_to_list() {\n\
           /usr/es/sbin/cluster/utilities/clvt query node\n\
           return 0\n\
  }\n\
  cmd_to_list"
  cmd_to_list_postfix = ""
  multi_select = ","
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "42,9"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Application Controller Start Script
#
# Script name used to start the application
#
# No default value
# This property is required
#
sm_cmd_opt:
  id_seq_num = "40"
  id = "clsa_gasa_modify_opts"
  disc_field_name = "start"
  name = "Application Controller Start Script"
  name_msg_file = "cluster.cat"
  name_msg_set = 43
  name_msg_id = 4
  op_type = ""
  entry_type = "t"
  entry_size = 256
```

```
  required = "+"
  prefix = "-R"
  cmd_to_list_mode = ""
  cmd_to_list = ""
  cmd_to_list_postfix = ""
  multi_select = ""
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "30,4"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""

# Application Controller Stop Script
#
# Script name used to start the application
#
# No default value
# This property is required
#
sm_cmd_opt:
  id_seq_num = "50"
  id = "clsa_gasa_modify_opts"
  disc_field_name = "stop"
  name = "Application Controller Stop Script"
  name_msg_file = "cluster.cat"
  name_msg_set = 43
  name_msg_id = 5
  op_type = ""
  entry_type = "t"
  entry_size = 256
  required = "+"
  prefix = "-O"
  cmd_to_list_mode = ""
  cmd_to_list = ""
  cmd_to_list_postfix = ""
  multi_select = ""
  value_index = 0
  disp_values = ""
  values_msg_file = ""
  values_msg_set = 0
  values_msg_id = 0
  aix_values = ""
  help_msg_id = "30,5"
  help_msg_loc = "cluster_hlp.cat"
  help_msg_base = ""
  help_msg_book = ""
# Service IP label
#
# The user must choose a service IP label to participate in the
# constructed PowerHA SystemMirror resource group
#
# The command to list cllsserviceips can be used by other Smart Assists
# to provide a ist of pre-configured service IP labels, and available
# labels that can be defined to PowerHA SystemMirror.
#
sm_cmd_opt:
  id_seq_num = "60"
  id = "clsa_gasa_modify_opts"
  disc_field_name = "ip"
  name = "Service IP Label"
  name_msg_file = "cluster.cat"
  name_msg_set = 43
```

```
name_msg_id = 6
op_type = "l"
entry_type = "t"
entry_size = 32
required = "+"
prefix = "-I"
cmd_to_list_mode = "1"
cmd_to_list = "/usr/es/sbin/cluster/sa/sbin/cllsserviceips"
cmd_to_list_postfix = ""
multi_select = "n"
value_index = 0
disp_values = ""
values_msg_file = ""
values_msg_set = 0
values_msg_id = 0
aix_values = ""
help_msg_id = "30,6"
help_msg_loc = "cluster_hlp.cat"
help_msg_base = ""
help_msg_book = ""
```

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as the customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Index

A

add application 40
add application_monitor 42
add cluster 26
add file_collection 47
add network 32
add node 27
add resource_group 34
add service_ip 38
APAR 15
API
  clvt 25
application controller class operations 40
application registration and query commands 23
application_monitor class operations 42

C

claddsa command 19
claddsaapp command 23
cllssaapp command 24
clquerysa command 21
clquerysaapp command 23
clrmsa command 22
clrmsaapp command 24
cluster class operations 26
clvt
  add application 40
  add application_monitor 42
  add cluster 26
  add file_collection 47
  add network 32
  add node 27
  add resource_group 34
  add service_ip 38
  application controller class operations 40
  application_monitor class operations 42
  cluster class operations 26
  delete application 41
  delete application_monitor 43
  delete cluster 26
  delete file_collection 48
  delete network 32
  delete node 27
  delete resource_group 34
  delete service_ip 39
  discover cluster 27
  file_collection class operations 47
  interface class operations 30
  modify interface 30
  modify location_dependency 46
  modify resource_group 35
  modify temporal_dependency 45
  network class operations 32
  node class operations 27
  offline node 29
  offline resource_group 38
  online node 29
  online resource_group 37
  query application 41
clvt *(continued)*
  query application_monitor 44
  query cluster 26
  query file_collection 48
  query interface 31
  query location_dependency 47
  query network 33
  query node 28
  query resource_group 36
  query service_ip 39
  query temporal_dependency 45
  resource group location_dependency class operations 46
  resource group temporal_dependency class operations 45
  resource_group class operations 34
  service_ip class operations 38
  sync cluster 26
clvt API 25
commands 18
  application registration and query 23
  claddsa 19
  claddsaapp 23
  cllssaapp 24
  clquerysa 21
  clquerysaapp 23
  clrmsa 22
  clrmsaapp 24
  registration and query 19
concepts 1
convenience routines 25

D

delete application 41
delete application_monitor 43
delete cluster 26
delete file_collection 48
delete network 32
delete node 27
delete resource_group 34
delete service_ip 39
developing panels 7
developing SMIT panels 7
directory structure 7
discover cluster 27
discovery database
  planning names 17
  planning values 17
discovery script
  example 9
disk space
  parameterized verification check 15

E

example
  discovery script 9

F

file
  parameterized verification check 15
file_collection class operations 47
fileset
  parameterized verification check 15
framework 3

G

group
  parameterized verification check 16

I

identifier 5
installing 6
interface class operations 30

M

manage menu 12
modify interface 30
modify location_dependency 46
modify resource_group 35
modify temporal_dependency 45

N

network class operations 32
node class operations 27

O

offline node 29
offline resource_group 38
online node 29
online resource_group 37

P

packaging 6
parameterized verification check 13, 14, 15
  APAR 15
  disk space 15
  file 15
  fileset 15
  group 16
  swap space 16
  user 16
planning names
  discovery database 17
planning values
  discovery database 17

Q

query application 41
query application_monitor 44
query cluster 26
query file_collection 48
query interface 31
query location_dependency 47
query network 33
query node 28
query resource_group 36
query service_ip 39
query temporal_dependency 45

R

registration and query commands 19
remove menu 12
requirements 2
resource group location_dependency class operations 46
resource group temporal_dependency class operations 45
resource_group class operations 34
routines
  SMIT 25

S

sample program 49
  add application instance 51
  commands 51
  deinstalling 50
  deleting application instance 55
  installing 49
  modify application instance 53
  overview 49
service_ip class operations 38
Smart Assist 7, 19, 23
  add application instance 51
  commands 18, 51
  component discovery 9
  concepts 1
  deinstalling 50
  deleting application instance 55
  framework 3
  identifier 5
  installing 6
  modify application instance 53
  overall flow 3
  overview 49
  packaging 6
  requirements 2
  sample program 49, 50, 51, 53, 55
    installing 49
  terms 1
SMIT 7, 12, 25
  add menu functionality 10
  add stanzas 55
  change/show 11
  modify stanzas 59
swap space
  parameterized verification check 16
sync cluster 26

T

terms
  application 1
  application instance 1
  discovery of applications 1

U

user
  parameterized verification check 16

V

verification
  custom 13
  parameterized 13, 14
    APAR 15
    disk space 15
    file 15
    fileset 15
    group 16
    swap space 16
    user 16

Printed in USA