Distributed Command Execution Manager

November 30, 2001

Distributed Command Execution Manager

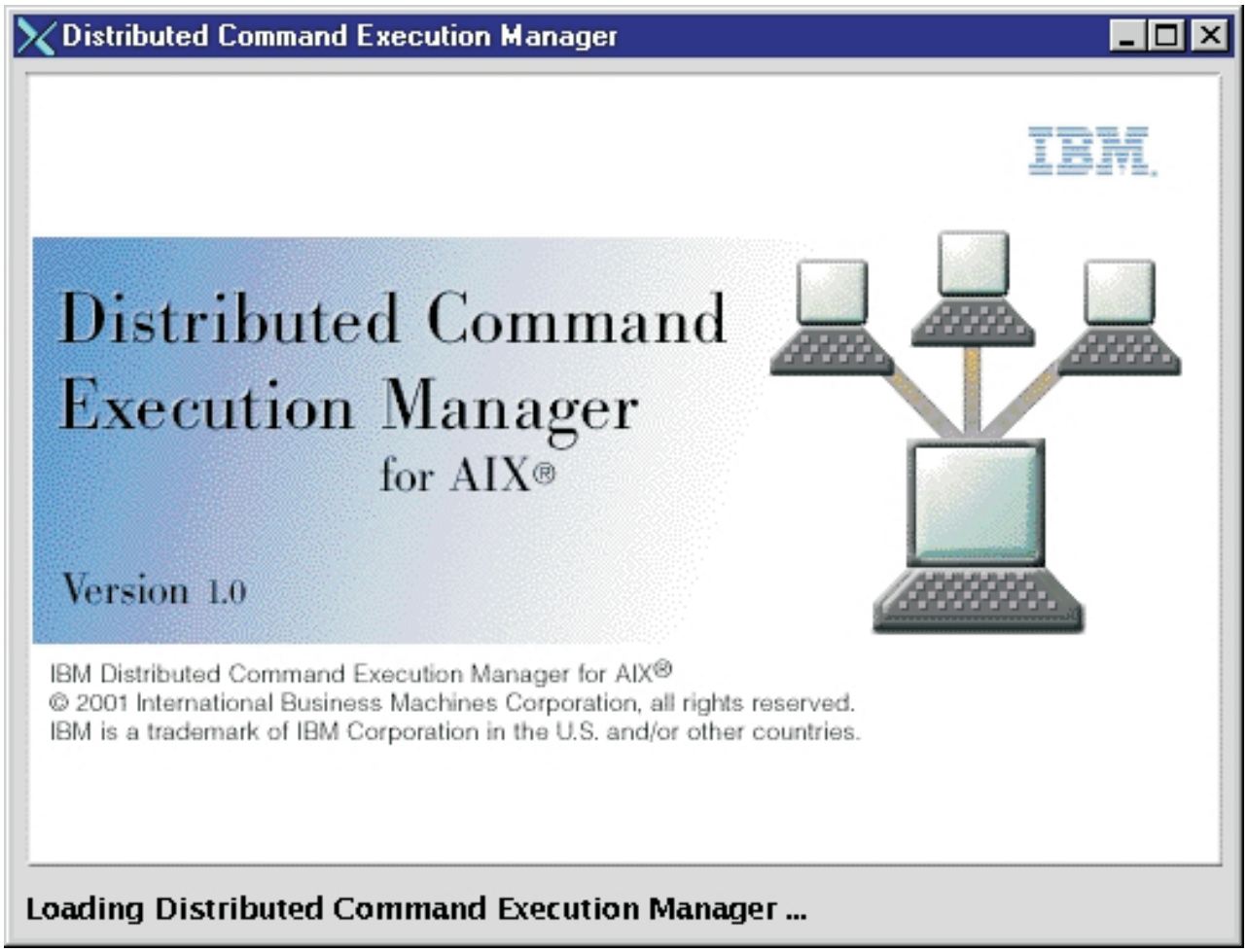
Before reading the information in this paper, read the general information in Appendix A: Notices.

Contents

Chapter 1. Distributed Command Execution Manager Overview

The Distributed Command Execution Manager (DCEM) provides a variety of services for a network of distributed machines. The DCEM graphical user interface allows you to construct command specifications for executing on multiple target machines, providing real-time status as commands are executed. You can enter the command definition, run-time options, and selected hosts and groups for a command specification, and you have the option of saving this command specification to use in the future. When you save a command specification, a Perl script is generated that you can run directly from a Linux or AIX command line. You can create and modify groups of hosts to use as targets for a command directly from DCEM. You can specify these groups by supplying host names for the group or by using dynamic queries on specific host attributes in a domain. DCEM also creates a log of all distributed command activity.

The DCEM startup window is shown in the following illustration.



1.1 Supported Platforms

You can use DCEM to manage systems running on the following operating systems:

- AIX 5L on PowerPC
- Red Hat Linux

1.2 Setting Up DCEM

DCEM uses IBM Cluster Systems Management (CSM). CSM uses the Resource Monitoring and Control (RMC) subsystem. Installing DCEM requires that you install and correctly configure both CSM and RMC. In particular, DCEM uses the CSM **dsh** command to run commands on the nodes. For **dsh** to work, you must set up security on each node.

If you choose the default remote shell **rsh** , you must add the management server host name to the /.rhosts file on the nodes that will be managed. To make the managed hosts visible to DCEM (through the Browse Hosts dialog), you must create node definitions for each of the nodes to be managed in the CSM database on the management server where DCEM will run. You can create these definitions when you install CSM. However, if these node definitions do not exist, you can create them by using the following CSM **createnode** command:

```
/opt/csm/bin/createnode -M host name
```

See the *CSM Technical Reference* for more information on the **createnode** command.

1.3 Starting DCEM

To start DCEM, type the following at the command line:

```
/opt/csm/dcem/bin/dcem
```

The DCEM command line options are:

```
/opt/csm/dcem/bin/dcem [-h | --help] [-V | --version] [-v | --verbose]
[-N | --groups [group,group,group,...]]
[-n | --hosts [host_name,host_name,host_name,...]]
[command_specification_name]
```

1.3.1 Command Syntax

Using the **dcem** command without options displays the Distributed Command Execution Manager dialog. From this dialog, you can create a new command specification or select from a list of saved command specifications.

Using the **dcem** command with the `command_specification_name` option causes DCEM to initialize the input fields in the main window with specified command data. `command_specification_name` refers to the name used to save a command specification in the DCEM dialog. To send the command defined in the command specification to the specified hosts or groups, click the **Run** button in the dialog. The Execution Progress dialog shows the progress of the executed commands. To reset DCEM to default values, click the **Defaults** button in the dialog.

You can also use the following flags:

- **-h | --help** writes the usage message for the **dcem** command to standard output.
- **-V | --version** writes version information to standard output.
- **-v | --verbose** runs the **dcem** command in debug mode and writes the command's verbose messages to standard output.
- **-N | --groups** [***group,group,group,...***] specifies the groups displayed in the Groups of hosts field of the DCEM dialog at startup. If you use this flag with a command specification name, the host names and groups that are stored as part of the command specification are ignored.
- **-n | --hosts** [***host_name,host_name,host_name,...***] specifies the hosts displayed in the **Host names** field of the DCEM dialog at startup. If you set this flag with a command specification name, the host names and groups that are stored as part of the command specification are ignored.

Example:

The following example specifies hosts and groups together with the `command_specification_name` option on the command line. Assume the **myCommand** command specification was saved with host names *h1, h2, h3*, and groups of hosts *g1, g2, g3*.

1. To run DCEM, type:

   ```
   dcem
   ```

2. To initialize the input fields with a specified command specification name and groups, type:

   ```
   dcem --groups g4,g5 myCommand
   ```

   This results in the following output in the following GUI fields:

   ```
   Host names:{empty}
   Groups of hosts: g4,g5
   ```

3. To initialize the input fields with specified command specification name, groups, and hosts, type:

   ```
   dcem --groups g4,g5 --hosts h4 myCommand
   ```

   This results in the following output in the following GUI fields:

   ```
   Host names: h4
   Groups of hosts: g4,g5
   ```

4. To display the version of DCEM that is running, type:

   ```
   dcem -V
   ```

Note: When you run DCEM from a remote host, run the **xhost +** command on that host from the machine you are using. On the machine running DCEM, run the **export** command, as follows:

```
export DISPLAY=IP address of the machine you are using
```

Chapter 2. Using Distributed Command Execution Manager

The following panels help you to define command specifications:

- General panel
- Options panel
- Groups panel
- Dynamic Groups panel

The General and Options panels provide an interface for creating new command specifications and modifying previously saved command specifications. The Groups and Dynamic Groups panels provide an interface for creating and modifying groups of host machines.

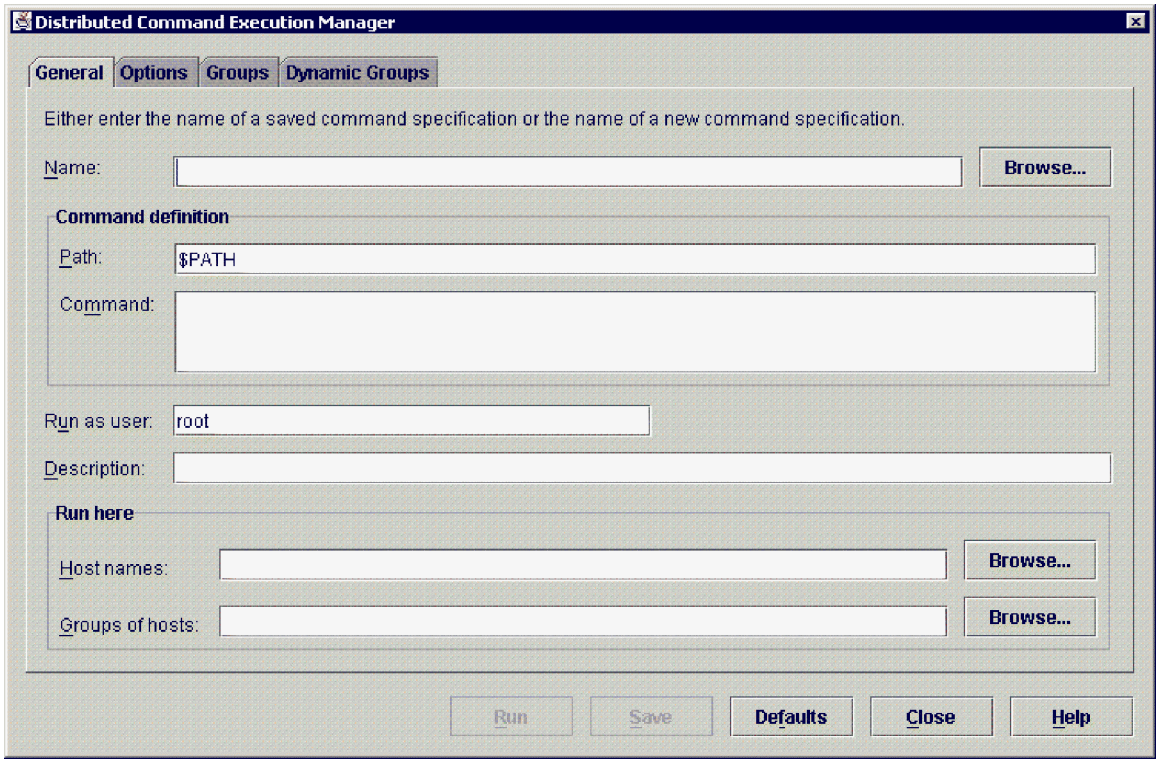
2.1 Creating Command Specifications

DCEM allows you to create, save, and edit command specifications, which reduces your time and effort when you repeatedly run the same command. A command specification consists of the following parts:

- The name of the command specification.
- A command definition including the path and command or inline script to run on remote host machines.
- The user name under which the command will run.
- A description of the command.
- A list of hosts or groups of hosts on which the command will run.
- Options for security, output streaming, and number of hosts on which to concurrently run the command.

2.1.1 General Panel

Use the following General panel to specify most of the information that is required to run commands on distributed hosts.



To create a command specification, you must provide, at a minimum, the following information in the text fields on this panel:

- **Name -** The name that identifies a command specification. When you create a new command specification, you must type a name for it in this field. The name field is not required for running a command specification, but is required when saving a specification.
- **Command -** The command or inline script to run, plus one or both of the following:
  - **Host Names -** The name of one or more hosts. You can type the name of any fully qualified host name as long as the host has a remote shell available, or type a list of fully qualified host names separated by commas or spaces. You can also select host names known to CSM from the Browse Hosts dialog, which displays when you click the **Browse** button next to the field.
  - **Groups of Hosts -** The name of one or more host groups. You can type a list of host groups separated by commas or spaces. You can also select host group names from the Browse Groups dialog, which displays when you click the **Browse** button next to the field. To use this selection dialog, you must first have created the groups of hosts. For more information, see Creating Groups of Hosts“2.4.1.1 Creating Groups of Hosts” on page 13.

The following fields, which contain default values, are also required, but are populated with default values:

- **Run as User -** The user name that the command will run under. By default, it is populated with the user name under which DCEM is running. You can edit this field. You must configure target machines to allow the user or machine under which DCEM is running to run as the user specified in this field. This configuration is specific to the remote shell used to run a command. (see Security Considerations and Remote Shells“3.3 Security Considerations and Remote Shells” on page 19)
- **Path -** The path that points to the actual location of the saved commands. The default value for this field is $PATH. You can edit this field.

  If you use the the default $PATH in this field, the application does not delegate the local $PATH to the target hosts when the command is run. Instead, it prepends `export PATH=$PATH` to the command, where the $PATH variable referred to is the one set on the target machine.

  To run commands found only in a specific directory, you can replace $PATH with that directory name. To guarantee that a directory is searched first, you can prepend the directory to $PATH, for example, /usr/bin:$PATH.

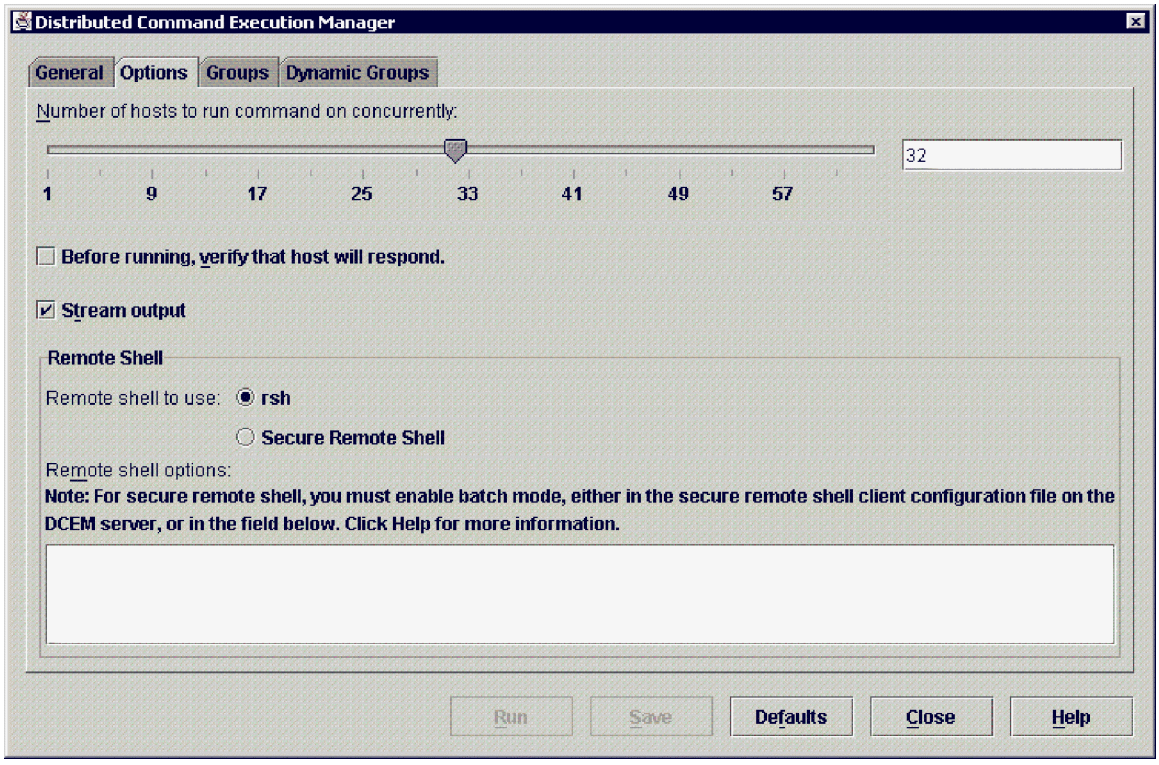
  If you leave the **Path** field empty, it is the same as having $PATH.

The following field is optional and has no default:

- **Description -** An optional text description of the command specification. This description also displays in the Browse Command Specifications dialog to help you locate a particular saved command specification.

2.1.2 Options Panel

When you create a command specification, you can also change the options for executing the commands on the distributed hosts. Use the following Options panel to view or change options.



The Options panel displays the following options:

- **Number of hosts to run commands on concurrently -** The slider and text box display the number of hosts on which the command will run at the same time. You can specify between 1 and 32 hosts, with a default value of 32. To change this value, either drag the slider to a new value or specify a value in the text box.
- **Before running, verify that hosts will respond** - Select this check box if you want to determine whether the hosts are online and responding before you run a command specification. By default, this check box is not selected, which allows commands to be sent to all hosts without checking whether they are available.
- **Stream Output -** Select this check box if you want to display output in the Execution Progress dialog as it is received. When this box is not selected, the output is collected and displayed only after command execution completes.
- **Remote Shell -** Displays the remote shell under which distributed commands will be run. The remote shell options for **rsh** listed in this section are for the AIX platform. On Linux, these options could be different. The default shell on AIX is **rsh**, and if the secure remote shell is not installed, **rsh** will be the only option available. You can enter options for either **rsh** or the secure remote shell in the text box provided. Enter the options as you would enter them on the command line.

  Because the DCEM application is not interactive, you must configure the secure remote shell to run in batch mode. If the secure remote shell is not configured properly and you are prompted for a password during authentication, the command that you attempted to run cannot execute. You must then click the **Stop** button at the bottom of the Execution Progress dialog to stop the execution.

  Options for **rsh** include the following:

  – **-f** causes DCE credentials to be forwarded to remote hosts. This option is valid only if the underlying **rsh** uses Kerberos authentication and you have valid Kerberos credentials. It will be ignored if Kerberos 5 is not the current authentication method, and authentication will fail if the current DCE credentials are not marked forwardable.

– **-F** causes the credentials on the remote system to be marked forwardable (allowing them to be passed to another remote system). This setting will also be ignored if Kerberos 5 is not the current authentication method.

DCEM supports a variety of secure remote shells and has been tested, to a limited basis, using OpenSSH, a secure remote shell. If a secure remote shell supports batch mode, you must enable the batch mode. You can do this either in the secure remote shell client configuration file on the CSM server or in the secure remote shell options in the DCEM Options panel. (If the secure remote shell is installed, you can select the **Secure Remote Shell** radio button, and in the field where the options for this selection display, you can type the flag that enables batch mode.) For detailed information about secure remote shell client configuration, see the specific secure remote shell documentation.

2.2 Saving a Command Specification

When you have completed entering your command specification information, click the **Save** button to save it as a script. Command specifications that you save are stored as Perl scripts (see Appendix C for an example of a saved command script). The saved script contains all the information on the DCEM General and Options panels. Saved command specifications are located in the following directory:

home/dcem/scripts/*script file name.pl*

home is the home directory of the user under whose name the distributed command is run. *script file name*.pl is the name of a Perl script file containing a saved command specification.

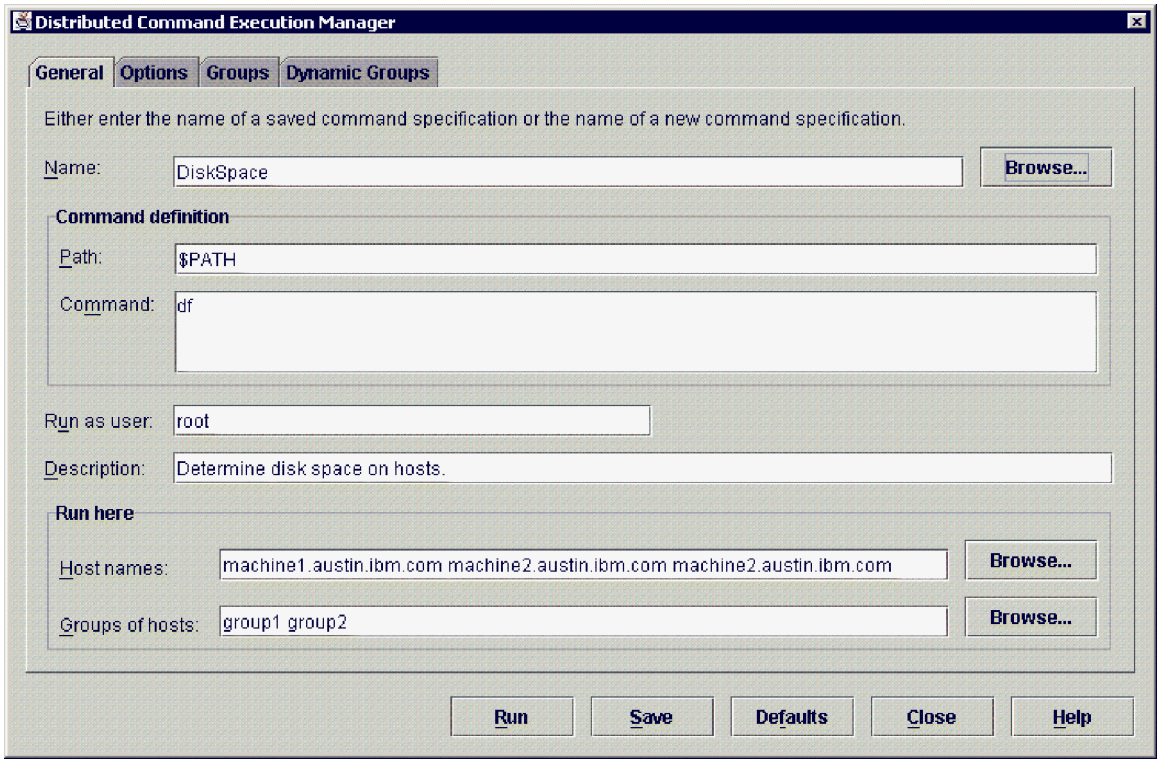
After you save a command specification, you can run it (see Running a Command on One or More Hosts“2.3 Running a Command on One or More Hosts”), view it, and select it from the Browse Command Specifications dialog.

2.3 Running a Command on One or More Hosts

You can run commands on multiple hosts using any of the following methods.

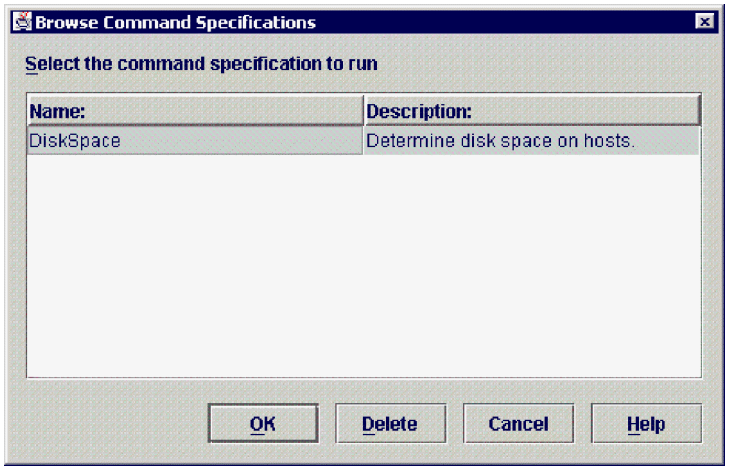
From the DCEM dialog

1. In the General panel, shown below, click the **Browse** button beside the **Name** text entry field.



This displays the Browse Command Specifications dialog.

2. In the Browse Command Specifications dialog, shown below, select a command specification from the list of existing command specifications, then click the **OK** button to load the selected command specification into the General and Options panels.



3. In the DCEM dialog, click the **Run** button to run the selected command specification on the specified hosts or groups of hosts.

From the DCEM dialog, you can also create a new command specification (see Creating Command Specifications“2.1 Creating Command Specifications” on page 5), then click the **Run** button to run the selected command specification on the specified hosts or groups of hosts.

Load the command specification from the command line

To load the command specification into the DCEM dialog directly from the command line, type the following:

```
/opt/csm/dcem/bin/dcem command_specification_name
```

Using the **dcem** command with the `command_specification_name` causes DCEM to initialize the input fields in the General and Options panels of the DCEM dialog with specified command data. You can then click the **Run** button to send the command to the specified hosts.

Run the command specification script on the command line

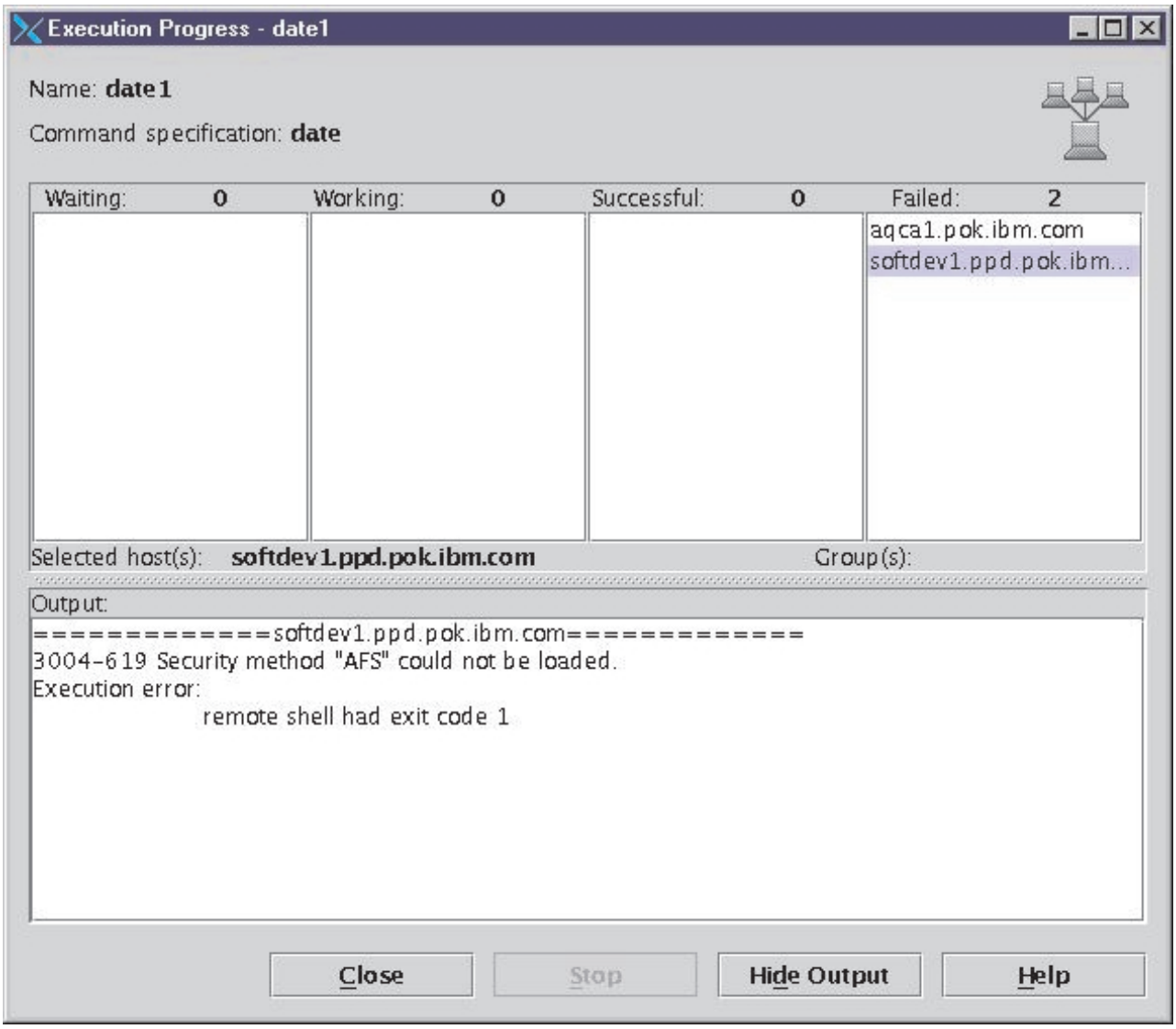
To run the command specification script directly on the command line, type the following:

```
user_home_directory/dcem/scripts/commandSpecificationName.pl
[-debug] [-non_interactive] [-format_output]
```

- **commandSpecificationName** - The name used to save a command specification using the DCEM dialog.
- **debug** - Verbose mode. Determines the actual execution string specified, for example, `/opt/csm/bin/dsh -f 11 -s -l root -N sysmgt-testbed "date"`.
- **non_interactive** - Does not prompt on the command line to run the command. This option is useful when invoking the command script from another script.
- **format_output** - Formats stdout output from all hosts. Output is grouped by host name.

2.3.1 Using the Execution Progress Dialog

After you click the **Run** button to run a valid command specification or command on valid hosts or groups of hosts, DCEM displays the following Execution Progress Dialog to show the status of the command execution on all of the hosts.



In this dialog, a series of lists show hosts on which command execution is in one of the following states:

- Waiting
- Working
- Successful
- Failed

The bottom of the window displays output from the command execution on selected hosts. To display output from a host, select its name from any of the lists. Multiple selections made from the Successful or Failed lists result in combined output in the output window. To view real-time output, select a host that is currently in the working state. The Waiting and Working lists do not support multiple selections.

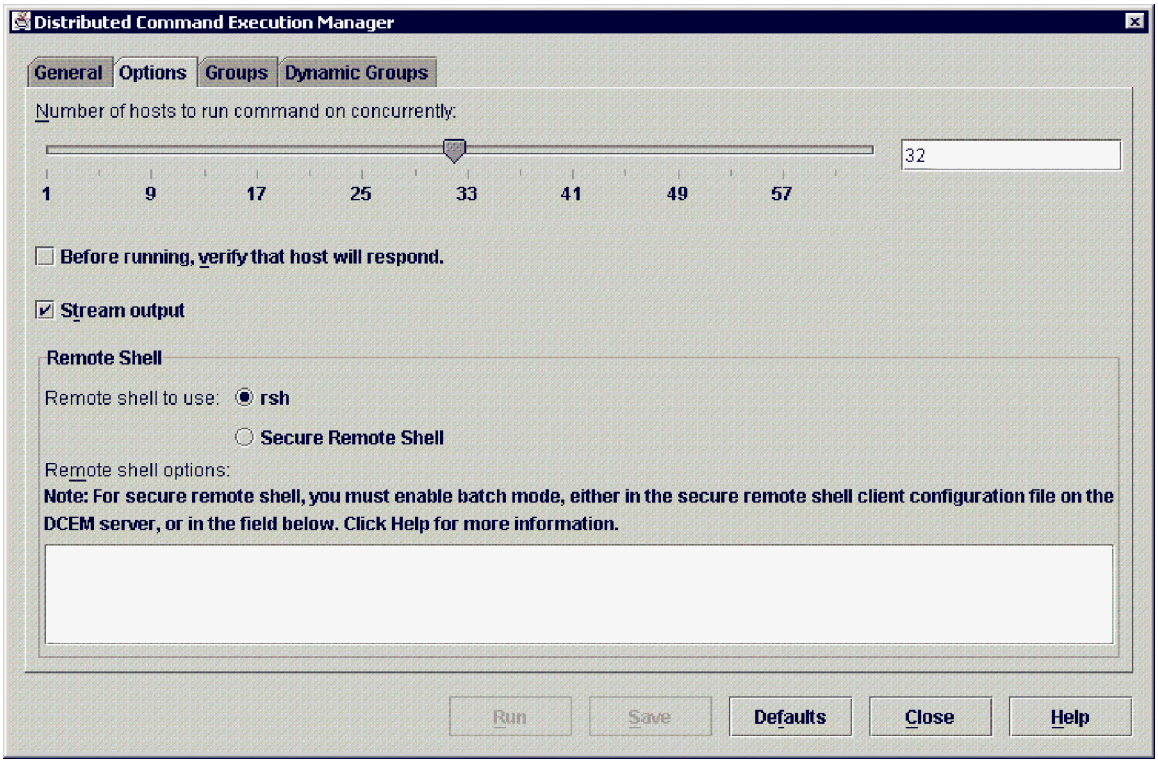
To stop the distributed command execution on all hosts, click the **Stop** button. If a command has not been completed on a host, command execution for that host is terminated. Hosts that have been stopped move to the Failed list. Command that have already completed may appear in either the Successful or Failed lists.

When the **Show Errors Only** check box is selected, the output area displays error messages for the selected hosts. **Stdout** messages are not displayed.

Selecting the **Close** button hides the Execution Progress Dialog, but does not stop execution on any hosts. If the DCEM dialog is closed before these hosts have completed execution, then these hosts will be stopped.

2.3.2 Fine-Tuning Run-Time Parameters

You can adjust several run-time options for your command using the DCEM Options panel, shown in the following illustration.



For example, if you are experiencing network problems and you want to improve performance, you can reduce the number of hosts on which the command specification will run at the same time. On the Options panel, the slider and the text box both display this value. The default is 32. Either of these can be modified with a new value between 1 and 32.

Note: When you change this value using the text field, the slider is only updated when the focus changes or the command is saved or run. It does not change as you type or when you press the **Enter** key.

You can also affect the amount of time it takes for commands to complete by selecting or deselecting the **Before running, verify that host will respond** check box. Selecting this check box allows you to invest the time to immediately check the host response. If there are problems, the wait time should be smaller than the minute typically taken for the remote shell command to time out.

You can change the default behavior of streaming the output (displaying it in the Execution Progress dialog as it is received) so that the output is collected and displayed for each host only after the command execution completes on that host and it is either in the successful or failed state.

You can also change the remote shell under which the commands run and specify options for that remote shell. The default remote shell is `rsh`.

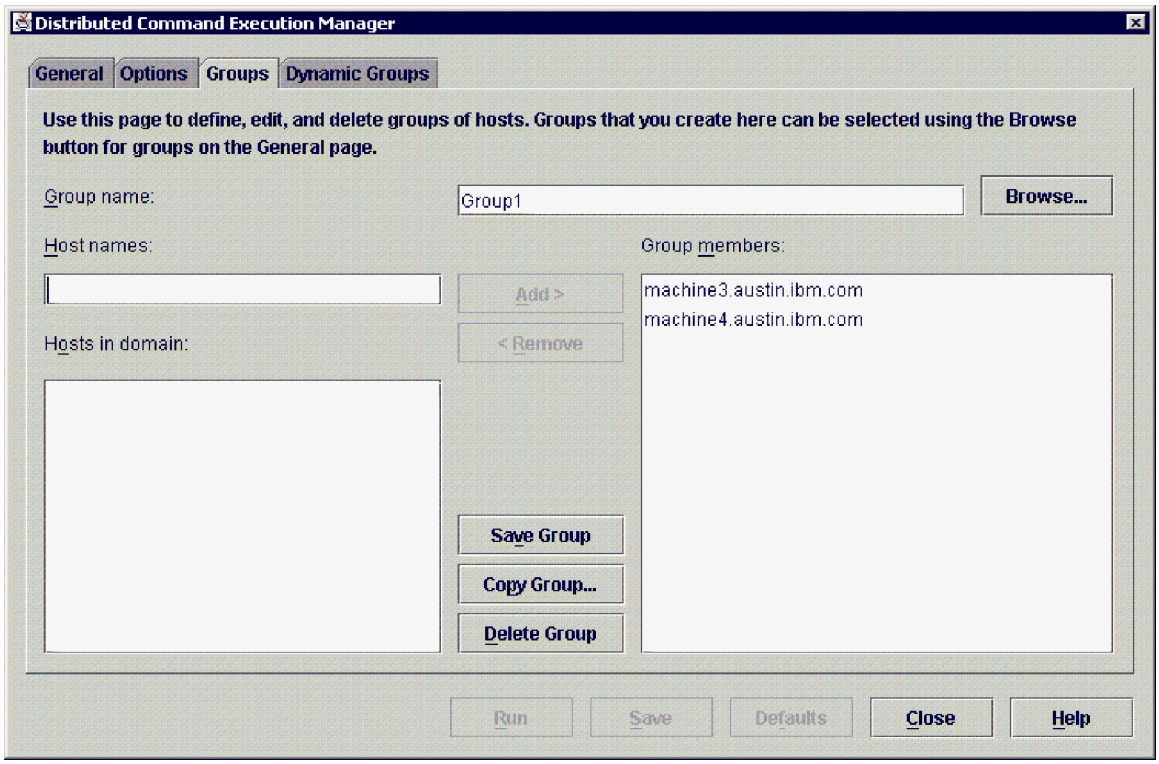
2.4 Working with Groups of Hosts

The ability to organize hosts into groups and to save these groups for later use can make it more convenient to run commands on the same groups of hosts repeatedly. Node groups can either be explicit lists of node host names, created by explicitly specifying each host, or they can be dynamic groups of hosts, created by specifying the desired selection criteria, such as ″Hostname like 'websrvr%'″. To create explicit groups of hosts for use in its command specifications, use the Groups panel in the DCEM dialog. To create dynamic host groups to use in its command specifications, use the Dynamic Groups panel.

Note: By default, the root user has the authority to create groups and a non-root user cannot create groups unless special permission is set for that user. The access authority is defined in the `/var/ct/cfg/ctrmc.acls` file. You can modify this file, then run the **refresh -s ctrmc** command to refresh the systems. For more information, see the *CSM for AIX Administration Guide*, Chapter 1: Cluster Systems Management Overview - Security Considerations.

2.4.1 Groups panel

Use the Groups panel to create explicit groups of hosts. This panel provides an interface for editing, deleting, and copying the groups that you have created. The **Group members** list box contains a list of hosts that are already in the group.



2.4.1.1 Creating Groups of Hosts

Use the Groups panel to create a group of hosts. To create a group of hosts using the DCEM Groups panel, do the following:

1. Type the name of the group you are creating in the **Group name** field.
2. To add a host to the group, either type its name in the **Host names** field or select the host from the **Hosts in domain** list box, then click the **Add >** button. The hosts listed in the **Hosts in domain** list box are those hosts defined in CSM that have CSM client code installed.

3. To add hosts to the group that are not defined in CSM, type the host name in the **Host names** field, then click the **Add >** button.
4. To create the group of hosts, click the **Save Group** button.

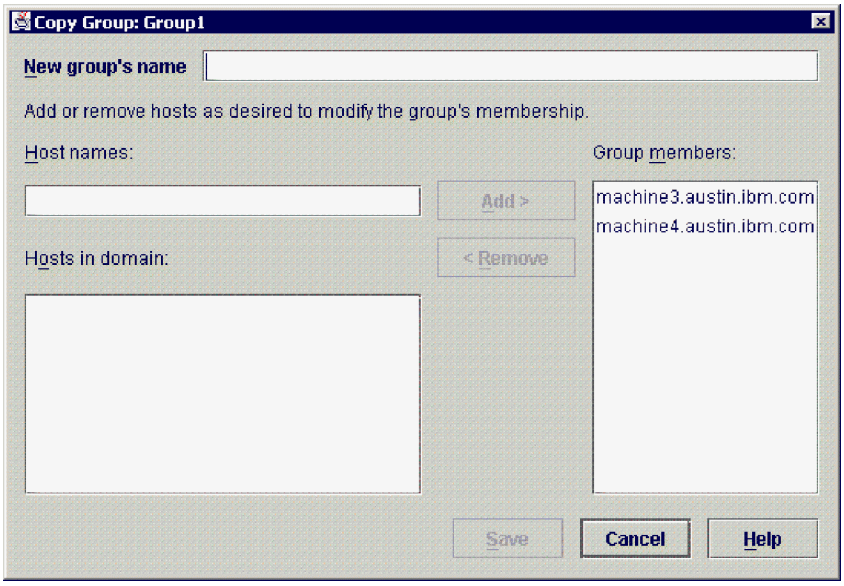
2.4.1.2 Editing Groups of Hosts

Use the Groups panel to edit an existing group of hosts. You can select group members to remove host names or add new host names. To edit a group of hosts, do the following:

1. In the **Groups** panel, click the **Browse** button beside the **Group name** box.
2. In the displayed **Browse Groups** dialog, select the group you want to edit.
3. To add a host to the selected group, either type a host name in the **Host names** field, or select a host from the **Hosts in domain** list box, then click the **Add >** button.
4. To delete a host from the selected group, select the hosts from the **Group members** list box, then click the **< Remove** button.
5. To save the changes, click the **Save Group** button.

2.4.1.3 Copying Groups of Hosts

Use the **Copy Group** dialog to copy an existing group of hosts. When you copy a group, you can also add new hosts to the group or remove hosts from the group before you save the copy.



To copy a group, do the following:

1. In the **Groups** panel, select the group you want to copy, then click the **Copy Group...** button.
2. In the **Copy Group** dialog, type the name of the new group in the **New group's name** field.
3. To add a host to the new group, either type a host name in the **Host names** field, or select a host from the **Hosts in domain** list box. To delete a host from the new group, select the host from the **Group members** list box, then click the **< Remove** button.
4. To save and copy the group, click the **Save** button.

2.4.1.4 Deleting Groups of Hosts

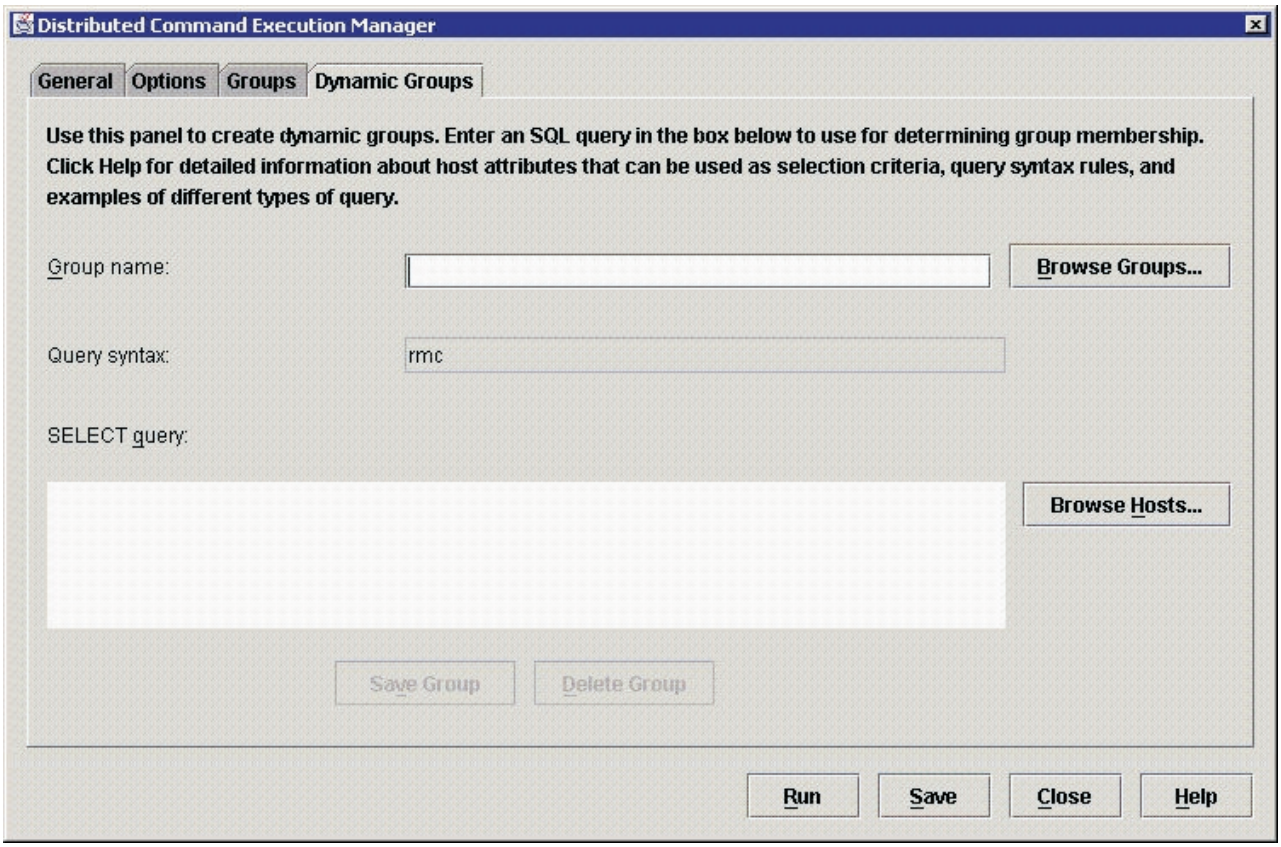
Use the Groups panel to delete existing groups of hosts. To delete a group of hosts, do the following:

1. In the **Groups** panel, select the group you want to delete, then click the **Delete Group** button.

2. To confirm, click the **Delete** button in the **Deleting Host Groups** confirmation dialog. To cancel, click the **Cancel** button.

2.4.2 Dynamic Groups panel

The Dynamic Groups panel, shown in the following illustration, allows you to create dynamic groups of hosts based on a select (SQL-like) string. This select string is used to search the hosts database to dynamically determine the list of hosts in the group.



2.4.2.1 Creating a Dynamic Group of Hosts

To create a dynamic group of hosts using the DCEM Dynamic Groups panel, do the following:

1. Type the name of the group that you are creating in the **Group Name** field. Alphanumeric characters are allowed.
2. Type the select string in the **SELECT Query** field. This is the select string that is used to dynamically determine which nodes belong to this group. The syntax for this select string is determined by the type of node database you are working with (as indicated in the **Query syntax** box on this panel). If your node database is a **file** type, use normal SQL syntax. If it is of **rmc** type, use the syntax described in the *IBM Cluster Systems Management for AIX Administration Guide*, in the chapter entitled ″Using Expressions″. To access this document, select **Download** from the (IBM Cluster Systems Management for AIX AlphaWorks web site) at http://9.6.22.30/tech/aixcsm and download the **am7TPadm.pdf** file.
3. To show the hosts that match your specified selection criteria after you specify a valid select string, click the **Browse Hosts** button.
4. To create the group of hosts, click the **Save Group** button.

2.4.2.2 Displaying Dynamic Group Host Members

To view host members of an existing dynamic group, do the following:

1. In the Dynamic Groups panel, click the **Browse Groups** button to display the **Browse Groups** dialog.
2. In the Browse Dynamic Groups dialog, select the dynamic group you want to view from the list of dynamic groups in the dialog. After you select the dynamic group, you are returned to the Dynamic Groups panel where the **Group name** field is filled in with the name of the selected group, and the **SELECT Query** field displays the chosen dynamic group's select string.
3. To view the hosts members of the selected dynamic group, click the **Browse Hosts** button beside the **SELECT Query** text field. The nodes that match the select string and that belong to the dynamic group are displayed.

   You can also use this feature to view hosts that satisfy the **SELECT Query** before actually creating a group. To do this, type the select string you want to use, then click on the **Browse Hosts** button. To view hosts matching a select string, you do not need to provide a group name.

2.4.2.3 Editing a Dynamic Group

Just as you cannot create a dynamic group by explicitly adding hosts to the group, you cannot edit a dynamic group by explicitly adding or removing hosts from the group. The only characteristic you can change about a dynamic group is its select string.

To edit an existing dynamic group, do the following:

1. In the Dynamic Groups panel, click the **Browse Groups** button to display the **Browse Groups** dialog.
2. In the Browse Groups dialog, select the dynamic group you want to edit.
3. In the **SELECT Query** field modify query. You can check the hosts members that will belong to the group, as defined by your select string, by clicking on the **Browse Hosts** button.
4. To save the selected dynamic group with its edited select string, click on the **Save Group** button.

2.4.2.4 Copying a Dynamic Group

To copy a dynamic group, do the following:

1. In the Dynamic Groups panel, click the **Browse Groups** button to display the **Browse Groups** dialog.
2. Select the dynamic group you want to copy by selecting a dynamic group from the list in the **Browse Dynamic Groups** dialog.
3. In the Dynamic Groups panel, edit the **Group name** text field with the desired name of the new group.
4. To save the new group click on the **Save Group** button to finish copying the group.

2.4.2.5 Deleting a Dynamic Group

To delete a dynamic group, do the following:

1. In the Dynamic Groups panel, click the **Browse Groups** button to display the **Browse Groups** dialog.
2. In the Browse Dynamic Groups dialog, select the dynamic group you want to delete.
3. In the Dynamic Groups panel, click on the **Delete Group** button to delete the chosen dynamic group.

2.4.2.6 Defining a SELECT Query String

You can write your own SQL queries to determine specific dynamic grouping criteria. By specifying the resource attribute in the SQL query, you can create a group on which a command can be run. These attributes are the node definitions in the IBM Cluster System Management database. Only persistent attributes can be used in a SELECT Query. To determine persistent attributes, type the following command at the command line:

```
lsrsrcdef -t -a p resource class or a resource | awk '{print $1}' | xargs -n3
```

The persistent attributes are listed in a three-column table. To list or test the **WHERE** clause of the SQL query string, type the following command at the command line:

```
lsnode -w "query_string"
```

To create a dynamic group, type:

```
nodegrp -w "query_string"
```

The *query_string* is what you specify in the SELECT query text area of the DCEM GUI in Dynamic Grouping panel.

Discovering Attributes Available for Queries

- To list all persistent attributes for resource IBM.ManagedNode, type the following command at the command line.

  ```
  lsrsrcdef -t -a p IBM.ManagedNode | awk '{print $1}' | xargs -n3
  ```

  The following output displays:

  ```
  Resource                program_name       Hostname
  Macaddr                 HWType             HWModel
  HWSerialNum             LParID             ConsolePortNum
  ConsoleServerName       HWControlPoint     SvcProcName
  OSType                  OSVersion          OSDistribution
  OSKernel                InstallDiskType    InstallDisk
  InstallMethod           UniversalId        ConsoleMethod
  ConsoleServerNumber     PowerMethod
  ```

- To list all available attributes of managed nodes, type the following command at the command line. Only persistent attributes can be used in creating a SQL query to determine a dynamic group.

  ```
  lsnode -Al
  ```

  Output similar to the following displays:

  ```
  Hostname = endive.austin.ibm.com

  OSVersion =

  UniversalId = 158933068
  .
  .
  .
  ConfigChanged = 0
  Status = 1
  OST
  ```

- To list dynamic attributes that belong to the IBM.ManagedNode object, type the following command at the command line.

  ```
  lsrsrc -a d IBM.ManagedNode
  ```

  Output similar to the following displays:

  ```
  Resource Dynamic Attributes for: IBM.ManagedNode

  resource 1:

  PowerStatus = 127

  Status = 1

  ConfigChanged = 0
  ```

Examples of Dynamic Grouping SQL Queries Usage

- To list the names of all the nodes types such as *host name c54* and exclude *c54n01* host names, type the following in the SELECT Query text area in the GUI:

  ```
  Hostname like 'c54%' && Hostname != 'c54n01.ppd.pok.ibm.com'
  ```

- To list the names of all the node types with a special power control, type the following in the SELECT Query text area in the GUI:

  ```
  PowerMethod == 'netfinity'
  ```

- To list all node names with the operating system type *linux*, type the following in the SELECT Query text area in the GUI:

  ```
  OSType like 'Linux%'
  ```

  The **OSType** attribute is defined during installation and is not a required attribute. If you did define the **OSType** attribute during the node installation, the above examples could be useful.

- To list all node names 'soft' or that have PowerMethod equal to 0, type:

  ```
  Hostname like 'soft%' || PowerMethod == '0'
  ```

For more detailed query and command syntax, see the *CSM Administration Guide*, Chapter 3: Using Expressions, and the *CSM Technical Reference*.

2.5 Command Output and Activity Logs

DCEM command output and activity are saved in log files. Log files are stored in the following directory:

home/dcem/log/*log file name*

home is the home directory of the user under whose name the distributed command is run. *log file name* is the name of the log file containing the **dcem** command activity. All DCEM command activity of failures and successes are saved in this log file.

The default log file name is `dcem1.log`. The default maximum size for a log file is 10M. When `dcem1.log` is full, it is renamed to `dcem2.log`. New log entries are always written to `dcem1.log`.

The following is an example of DCEM log file contents.

```
TIME:      Sep 20  18:55:57.076
INFO:      Command Name:listing
Command: ls -l
Successful Machines: wsm14 wsm12 wsm06 wsm04 wsm15 wsm03
Failed Machines: wsm01 wsm00


TIME:      Oct 08  11:32:10.868
INFO:      Command Name:DiskSpace
Command: df
Successful Machines: endive.austin.ibm.com
Failed Machines: westwing.austin.ibm.com
```

Chapter 3. Diagnosing Problems with Distributed Command Execution Manager

DCEM uses IBM Cluster Systems Management (CSM), which, in turn, uses several other tools. Understanding this relationship can be helpful in diagnosing problems.

3.1 Problems Due to Insufficient Setup of Underlying Subsystems

The underlying CSM uses the Resource Monitoring and Control (RMC) subsystem to monitor and communicate with all nodes. If you are experiencing problems communicating with managed nodes, verify that RMC is running on each node, and that the security access and control list (ACL) file has been set up to allow the nodes to communicate with the management server.

DCEM uses the CSM **dsh** command to run commands on the nodes. In order for the **dsh** command to work, security needs to be set up on each node in such a way that **dsh** is allowed to run commands on that node. The security setup is dependent upon the type of remote shell you are using. The default remote shell is **rsh**, and to set up security on each node to allow dsh to run commands on that node (using **rsh**), you must add the management server host name to the /.rhosts file on the nodes that will be managed nodes. For example, if you want to run commands as root on machine2 from machine1, to the /.rhosts file on machine2, you would have to add the line `machine1 root`.

To verify the successful installation of CSM, list the active nodes by running the **lsnode -p** command and verify that **dsh** is working by running the **dsh -a date** command. For for information, see the *AIX CSM Planning and Installation Guide*.

3.2 Interactive Commands and GUI applications

The CSM **dsh** command does not support the execution of interactive commands. Therefore, attempting to run an interactive command (one that requires input from standard in) from DCEM will not work.

To run an XWindows GUI application from DCEM, make sure that the DISPLAY variable is first set to your system's DISPLAY address, so that the GUI will display on your system. (For example, in the General panel, command area, you could first export the DISPLAY variable to your display's address prior to issuing your command name.) If you run a GUI application correctly from DCEM, the application will remain in the ″Working″ state until you choose to exit the GUI.

3.3 Security Considerations and Remote Shells

DCEM takes in the same underlying security considerations as the CSM **dsh** command. You can use any underlying remote shell, but it is the system administrator's responsibility to configure and enable remote shell access. DCEM uses the CSM **dsh** command, which uses the underlying **rsh** security protocol by default. For more information about security considerations for **dsh** and preparing for **dsh** and configuring the remote shell, see the *CSM AIX Planning and Installation Guide*.

3.4 Diagnostic Information

All DCEM command activity of failures and successes are saved in log files to use later if you have to diagnose problems. These log files are stored in the following directory:

home/dcem/log/*log file name*

To see more detail on the actual underlying CSM command execution string specified as a result of running your created command specification, run the Perl script (outside of DCEM) in debug mode and directly from your AIX command line, as follows:

commandSpecificationName.pl -debug

3.5 Diagnostic Information in Related Publications

Diagnosing DCEM problems relies heavily on being able to diagnose problems in the underlying CSM and RMC subsystems. For more information on diagnosing problems in DCEM's underlying subsystems, refer to the *IBM Cluster Systems Management for AIX Administration Guide*, in the chapter entitled ″Diagnostic Information″. To access this document, select **Download** from the (IBM Cluster Systems Management for AIX AlphaWorks web site) at http://9.6.22.30/tech/aixcsm and download the **am7TPadm.pdf** file.

Appendix A: Notices

This document was produced in the United States. IBM may not offer the products, programs, services or features discussed herein in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the products, programs, services, and features available in your area. Any reference to an IBM product, program,service or feature is not intended to state or imply that only IBM's product, program, service or feature may be used. Any functionally equivalent product, program, service or feature that does not infringe on any of IBM's intellectual property rights may be used instead of the IBM product, program, service or feature.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquires, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The information contained in this document has not been submitted to any formal IBM test and is distributed ″AS IS″. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. The use of this information or the implementation of any techniques described herein is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. Customers attempting to adapt these techniques to their own environments do so at their own risk.

IBM is not responsible for printing errors in this publication that result in pricing or information inaccuracies.

The information contained in this document represents the current views of IBM on the issues discussed as of the date of publication. IBM cannot guarantee the accuracy of any information presented after the date of publication.

Any performance data contained in this document was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally available systems. Some measurements quoted in this document may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries: IBM, AlphaWorks, AIX, AIX 5L. A full list of U.S. trademarks owned by IBM can be found at http://iplswww.nas.ibm.com/wpts/trademarks/trademar.htm Other company, product and service names may be trademarks or service marks of others.

Appendix B: DCEM Command Man Page

Name

dcem provides a graphical user interface (GUI) that allows you to run a command or script on multiple distributed nodes on a network at the same time.

Synopsis

dcem [**-h** | **--help**] [**-V** | **--version**] [**-v** | **--verbose**] [**-N** | **--group** [*group,group,group,...*]] [**-n** | **--hosts** [*host_name,host_name,host_name,...*]] [*command_specification_name*]

Description

Distributed Command Execution Manager (DCEM) provides a GUI that allows you to run a command or script on multiple distributed machines on a network at the same time. You can specify a collection of individual nodes, or you can create groups of nodes and save them to use again. DCEM provides real-time command execution status on the individual nodes that you specify, showing them in a waiting, working, successful, or failed state. It helps you to create, save, and edit command specifications, and it also creates a log of all distributed command activity. DCEM provides the capability of saving command specifications as PERL scripts. These saved command specifications can be executed at the command line. The command specifications are saved under the user's home directory **/home/dcem/scripts**.

The *command_specification_name* parameter initializes the input fields with the specified command. This is the name that is used when the command is saved in the DCEM GUI.

For the first release, the **dsh** command provides the underlying function.

Options

-h | --help	Writes the command's usage message to standard output.
-V | --version	Writes version information to standard output.
-v | --verbose	Runs in debug mode and writes the command's verbose messages to standard output.
-N | --groups [***group, group,group,...***]	Specifies the name of a group of hosts displayed in the **Groups of hosts** field in the DCEM dialog. If you use this option with the command specification name, the host names and groups that were saved with the command are ignored.
-n | --hosts [***host_name,host_name,host_name,...***]	Specifies the name of the hosts displayed in the **Host names** field of the DCEM dialog. If you use this option with the command specification name, the host names and groups that were saved with the command are ignored.

Examples

The following are examples for specifying hosts and groups together with the *command_specification_name* parameter on the command line. Assume the **myCommand** command was saved with the host names **h1**, **h2**, **h3** and groups of hosts **g1**, **g2**, **g3**.

- To run DCEM, type:

  ```
  dcem
  ```

- To initialize the input fields with specified command and groups, type:

  ```
  dcem --groups g4,g5 myCommand
  ```

This results in the following output in the following GUI fields:

```
Host names: {empty}
Groups of hosts: g4,g5
```

- To initialize the input fields with specified command name, groups, and hosts, type:

  ```
  dcem --groups g4,g5 --hosts h4 myCommand
  ```

  This results in the following output in the following GUI fields:

  ```
  Host names: h4
  Groups of hosts: g4,g5
  ```

- To display the version of DCEM that is running, type:

  ```
  dcem -V
  ```

Files

/home/dcem/scripts/*script_filename.pl*	Location of the DCEM command specification scripts.
/home/dcem/logs/*log_filename*	Location of DCEM log files

See Also

The **dsh** command

Appendix C: Example of a saved command script

```
#!/usr/bin/perl -w

#############################################################################
#
# Licensed Materials - Property of IBM
#
# (C) COPYRIGHT International Business Machines Corp. 1994,2001
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
#############################################################################

#############################################################################
#
# Example perl file -
# Run via:
#   perl <this-perl-script.pl>  [-debug] [-non_interactive]
# E.g.
#   perl listusers.pl
#   perl listusers.pl -debug
#   perl listusers.pl -non_interactive
#
# Author : Generated by Distributed Command Execution Manager
#
#############################################################################

#############################################################################
# perl information
#
#############################################################################

$| = 1;           # Flush output buffer
require 5.003;    # need this version of Perl or newer
use English;      # use English names, not cryptic ones
use FileHandle;   # use FileHandles instead of open(),close()
use Carp;         # get standard error / warning messages
use strict;       # force disciplined use of variables

#############################################################################
# GLOBAL VARIABLES AND CONSTANTS
#
#############################################################################

my ($TRUE)                              = "TRUE";
my ($FALSE)                             = "FALSE";

#
----------------------------------------------------------------------
# Command Environment Variables
#
----------------------------------------------------------------------

my (@HOSTS)                             =
('b905em17.austin.ibm.com');
my (@GROUPS)                            = ('Group1','Group2');

#
----------------------------------------------------------------------
# Command Specification
#
----------------------------------------------------------------------
```

```
my ($CMD_SPECIFICATION)=>'END_CMD_SPECIFICATION'
ls -l; whoami; pwd; ls -l
END_CMD_SPECIFICATION
;

#
#-------------------------------------------------------------------------
# Script options and user default settings
# NOTE: You must add any new options to
# the OPTION_FLAGS array.
#
#-------------------------------------------------------------------------

my ($DEBUG_FLAG)               = "-debug";
my ($LAUNCH_GUI_FLAG)          = "-gui";
my ($FORMAT_OUTPUT_FLAG)       = "-format_output";
my ($PROMPT_USER_FLAG)         = "-non_interactive";
my (@OPTION_FLAGS)             = ($DEBUG_FLAG,
                                  $LAUNCH_GUI_FLAG,
                                  $FORMAT_OUTPUT_FLAG,
                                  $PROMPT_USER_FLAG);

my ($DEBUG)                    = $FALSE;
my ($LAUNCH_GUI)               = $FALSE;
my ($FORMAT_OUTPUT)            = $FALSE;
my ($PROMPT_USER)              = $TRUE;

#
#-------------------------------------------------------------------------
# Csm Distributed Services
#
#-------------------------------------------------------------------------

my ($DISTRIB_SERVICE)                     = "/opt/csm/bin/dsh";
my ($DISTRIB_POST_PROCESSING_COMMAND)     = "/opt/csm/bin/dshbak";
my ($DISTRIB_DEFAULT_REMOTE_SHELL)        = "rsh";

#
#-------------------------------------------------------------------------
#      Dsh Options
#
#-------------------------------------------------------------------------

my ($DISTRIB_HOST_OPTION)                 = "-n";
my ($DISTRIB_GROUP_OPTION)                = "-N";
my ($DISTRIB_FANOUT_OPTION)               = "-f";
my ($DISTRIB_STREAMING_OPTION)            = "-s";
my ($DISTRIB_VERIFY_HOSTS_OPTION)         = "-v";
my ($DISTRIB_USER_OPTION)                 = "-l";
my ($DISTRIB_REMOTE_SHELL_OPTIONS_OPTION) = "-o";
my ($DISTRIB_REMOTE_SHELL_PATH_OPTION)    = "-r";

#
#-------------------------------------------------------------------------
# Additional Csm Command Environment Variables
#
#-------------------------------------------------------------------------

my ($FANOUT)                               = 64;
my ($STREAMING)                            = $TRUE;
my ($VERIFY_HOSTS)                         = $FALSE;
my ($USER)                                 = "root";
my ($REMOTE_SHELL_OPTIONS)                 = "";
my ($REMOTE_SHELL)                         = "rsh";
my ($COMMAND_PATH)                         = '$PATH';

#
```

```
------------------------------------------------------------------------

#############################################################################
# Sub Functions
#############################################################################

#############################################################################
# This sub-function displays the string passed to it.
# This sub-function should be used only for debug messages.
#
# @param the messages string to be displayed
#
#############################################################################

sub debug_message ($)
{
    if ($DEBUG eq $TRUE)
    {
        display_message (@_);
    }
}

#############################################################################
# This sub-function displays the string passed to it.
# This sub-function should be used to convey information to
# users
#
# @param the messages string to be displayed
#
#############################################################################

sub display_message ($)
{
    my ($str) = @_;
    print "$str";
}

#############################################################################
# This sub-function executes the command.
#
#############################################################################

sub run_distributed_command_line ()
{
    debug_message ("Enter sub-function run_distributed_command()...\n");

   my (@execution_string);
    my ($line) = "";

   # Get the arguments

   # construct the execution string based on the parameters
    @execution_string = build_execution_string();

   # run the command
    debug_message ("Running the command: @execution_string \n\n");

   my (@output);
    my ($current_pid) = fork;
    if ($current_pid == 0) {
        @output = exec (@execution_string);
    }
    elsif ($current_pid) {
        debug_message("In parent process, before wait.\n");
        my ($child_pid) = wait;
        debug_message("In parent process, after wait. Child pid was
$child_pid.\n");
```

```
    }
    else {
        die "fork error: $!\n";
    }

   foreach $line (@output)
    {
        display_message ($line);
    }

   debug_message ("\nLeave sub-function run_distributed_command().\n");
    return (@output);
}

#############################################################################
# This sub-function invokes the
# Distributed Command Execution Manager GUI
#
#############################################################################

sub run_distributed_command_gui ()
{
    debug_message ("Enter sub-function run_distributed_command_gui()...
\n");

   my ($cmd_name) = $0;
    $cmd_name =~ s/\.pl$//;
    debug_message( "Command name to load is $cmd_name\n");

   '/opt/csm/dcem/bin/dcem -command $cmd_name';

   debug_message ("Leave sub-function
run_distributed_command_gui().\n");
}

#############################################################################
# This sub-function gets the path to the selected remote shell.
#
#-- This is generated by the printBuildCommandLineFunction() method
#############################################################################

sub get_remote_shell_path ()
{
    debug_message ("Enter sub-function get_remote_shell_path()...");

   my ($execution_string) = "ksh -c \"which $REMOTE_SHELL\"";
    my ($line);

   # run the command
    debug_message ("Running the which $REMOTE_SHELL command:
$execution_string \n\n");

   my (@output) = '$execution_string';

   $line = $output[0];
    my (@splitLine) = split(' ', $line);
    my ($path) = $splitLine[0];

   if ($line =~ /$REMOTE_SHELL$/)
    {
        debug_message ("Found remote shell path $path\n");
        return $path;
    }
    else
    {
        display_message ("The remote shell $REMOTE_SHELL was not found.
Dsh
```

```
will use the default $DISTRIB_DEFAULT_REMOTE_SHELL remote shell when
executing this command.\n");
        return "";
    }
}

#########################################################################
# This sub-function contructs the complete execution string.
#
#########################################################################

sub build_execution_string ()
{
    debug_message ("Enter sub-function build_execution_string()...\n");

   my ($i) = 0;
    my ($cmd_path) = "";
    my (@execution_string);

   $execution_string[$i] = $DISTRIB_SERVICE;
    $execution_string[++$i] = $DISTRIB_FANOUT_OPTION;
    $execution_string[++$i] = $FANOUT;
    debug_message ("Execution string is: @execution_string\n");

   if (($STREAMING eq $TRUE) && ($FORMAT_OUTPUT eq $FALSE))
    {
        $execution_string[++$i] = $DISTRIB_STREAMING_OPTION;
        debug_message ("Execution string is: @execution_string\n");
    }

   $execution_string[++$i] = $DISTRIB_USER_OPTION;
    $execution_string[++$i] = $USER;
    debug_message ("Execution string is: @execution_string\n");

   if ($REMOTE_SHELL ne $DISTRIB_DEFAULT_REMOTE_SHELL)
    {
        my ($remote_shell_path) = get_remote_shell_path();
        if ($remote_shell_path ne "")
        {
            $execution_string[++$i] = $DISTRIB_REMOTE_SHELL_PATH_OPTION;
            $execution_string[++$i] = $remote_shell_path;
            debug_message ("Execution string is: @execution_string\n");

           $execution_string[++$i] =
$DISTRIB_REMOTE_SHELL_OPTIONS_OPTION;
            $execution_string[++$i] = $REMOTE_SHELL_OPTIONS;
            debug_message ("Execution string is: @execution_string\n");
        }
    }
    elsif ($REMOTE_SHELL_OPTIONS ne "")
    {
        $execution_string[++$i] = $DISTRIB_REMOTE_SHELL_OPTIONS_OPTION;
        $execution_string[++$i] = $REMOTE_SHELL_OPTIONS;
        debug_message ("Execution string is: @execution_string\n");
    }

   if (@GROUPS)
    {
        $execution_string[++$i] = $DISTRIB_GROUP_OPTION;
        $execution_string[++$i] = join(",", @GROUPS);
        debug_message ("Execution string is: @execution_string\n");
    }

   if ($VERIFY_HOSTS eq $TRUE)
    {
        $execution_string[++$i] = $DISTRIB_VERIFY_HOSTS_OPTION;
        debug_message ("Execution string is: @execution_string\n");
```

```
    }

   if (@HOSTS)
    {
        $execution_string[++$i] = $DISTRIB_HOST_OPTION;
        $execution_string[++$i] = join(",", @HOSTS);
        debug_message ("Execution string is: @execution_string\n");
    }

   if ($COMMAND_PATH ne '')
    {
        $cmd_path = "export PATH=$COMMAND_PATH;";
        debug_message ("Command path string is: $cmd_path\n");
    }

   $execution_string[++$i] = join(" ", $cmd_path, $CMD_SPECIFICATION);
    debug_message ("Execution string is: @execution_string\n");

   if ($FORMAT_OUTPUT eq $TRUE)
    {
        $execution_string[++$i] = join("", " | ",
$DISTRIB_POST_PROCESSING_COMMAND);
        debug_message ("Execution string is: @execution_string\n");
    }

   debug_message ("Leave sub-function build_execution_string().\n");
    return (@execution_string);
}

#########################################################################
# This sub-function asks the user whether the program should
# continue or not.
#
# @param cmd_spec - the command specification
# @param hosts    - the host machines to run the command on
# @param groups   - the groups to run the command on
#
#########################################################################

sub confirm_command_execution ($$$)
{
    debug_message ("Enter sub-function
confirm_command_execution()...\n");

   my ($cmd_spec, $hosts_ref, $groups_ref) = @_;
    my (@hosts) = @$hosts_ref;
    my (@groups) = @$groups_ref;
    my ($host);
    my ($group);
    my ($reply) = "";

   display_message("The command \"$cmd_spec\" ");

   if ((scalar(@hosts) == 0) && (scalar(@groups) == 0))
    {
        display_message("has no targets specified.\n");
        return($FALSE);
    }

   display_message ("is about to be executed on the following ");

   if (scalar(@hosts))
    {
        display_message( "hosts:\n\t");

       foreach $host (@hosts)
        {
```

```
            display_message ("$host ");
        }
        display_message ("\n");

       if (scalar(@groups))
        {
            display_message ("and ");
        }
    }

   if (scalar(@groups))
    {
        display_message ("groups:\n\t");

       foreach $group (@groups)
        {
            display_message ("$group ");
        }
        display_message ("\n");
    }

   while (defined($reply) && $reply !~ /[yYnN]/ )
    {
        display_message ("Do you wish to continue (y/n)?: ");
        $reply = <STDIN>;
        chop ($reply);
    }

   debug_message ("Leave sub-function confirm_command_execution().\n");

   # Check the reply to determine whether to continue
    if ($reply =~ /[yY]/)
    {
        return($TRUE);
    }
    else
    {
        return ($FALSE);
    }
}

#############################################################################
# This sub-function exits the program with an appropriate
# exit code.
#
# @param
#
#############################################################################

sub exit_program ($$)
{
    debug_message ("Enter sub-function exit_program()...\n");

   my ($msg, $exit_code) = @_;

   display_message ($msg);

   debug_message ("Leave sub-function exit_program().\n");
    debug_message ("Exiting program with exit_code: $exit_code\n");
    exit ($exit_code);
}

#############################################################################
# This sub-function display the usage message for this command
#
# @param
#
```

```
#########################################################################

sub usage ($)
{
    debug_message ("Enter sub-function usage()...\n");

   my ($bad_option) = @_;
    display_message ($bad_option);

   display_message ("Usage: perl $0 [$DEBUG_FLAG] [$PROMPT_USER_FLAG]
\n\n");

   display_message (" $DEBUG_FLAG\t\t\t- displays debug messages\n");
    display_message (" $PROMPT_USER_FLAG\t- does not prompt user for
input\n");

   exit_program("", 1);

   debug_message ("Leave sub-function usage().\n");
}

#########################################################################
# This function logs entries into a file whose name is
# provided as the input argument.
#
# @param - log file name
#
#########################################################################

sub generate_log_entries
{
    debug_message ("Enter sub-function generate_log_entries()...\n");
    '/opt/csm/dcem/bin/dLogMgr -s \"@_\"';

   debug_message ("Leave sub-function generate_log_entries().\n");
}

#########################################################################
# This function mails status reports after the command is
# executed
#
# @param - email addresses
#
#########################################################################

sub mail_report ($)
{
    debug_message ("Enter sub-function mail_report()...\n");

   debug_message ("Leave sub-function mail_report().\n");
}

#########################################################################
# This sub-function check all the options, in the
# @ARGV array.
# Flags are assumed to begin with a '-' (dash or minus sign).
#
#########################################################################

sub check_options ()
{
    debug_message ("Enter sub-function check _options ()...\n");

   my ($CMD_OPTION) = "";
    my ($TMP_CMD_OPTION) = "";
    my ($OPTION_FLAG) = "";
```

```
   foreach $CMD_OPTION (@ARGV)
    {
        $TMP_CMD_OPTION = "";

       # Check for incomplete and/or ambiguous options
        foreach $OPTION_FLAG (@OPTION_FLAGS)
        {
            if (index ($OPTION_FLAG, $CMD_OPTION) ==  0)
            {
                if ($TMP_CMD_OPTION eq "")
                {
                    $TMP_CMD_OPTION = $OPTION_FLAG;
                }
                else
                {
                    $TMP_CMD_OPTION = "AMBIGUOUS"; #ambiguous
                    last;
                }
            }
        }

       if ($TMP_CMD_OPTION eq $PROMPT_USER_FLAG)
        {
            debug_message ("Setting PROMPT_USER to FALSE.\n");
            $PROMPT_USER = $FALSE;
        }
        elsif ($TMP_CMD_OPTION eq $LAUNCH_GUI_FLAG)
        {
            debug_message ("Setting LAUNCH_GUI to TRUE.\n");
            $LAUNCH_GUI = $TRUE;
        }
        elsif ($TMP_CMD_OPTION eq $FORMAT_OUTPUT_FLAG)
        {
            debug_message ("Setting FORMAT_OUTPUT to TRUE.\n");
            $FORMAT_OUTPUT = $TRUE;
        }
        elsif ($TMP_CMD_OPTION eq $DEBUG_FLAG)
        {
            $DEBUG = $TRUE;
            debug_message ("Setting DEBUG to TRUE.\n");
        }
        elsif ($TMP_CMD_OPTION eq "AMBIGUOUS")
        {
            usage ("Ambiguous option: $CMD_OPTION\n\n");
        }
        else
        {
            usage ("Error! Bad option: $CMD_OPTION\n\n");
        }
    }

   debug_message ("Leave sub-function check _options ().\n");
}

#########################################################################
# This sub-function invokes all other subfunctions and is
# responsible for executing the command.
#
#########################################################################

sub main_driver ()
{
    # Check if any command line arguments have been passed in
    check_options ();

   debug_message ("Enter function main_driver()...\n");
```

```
   my ($host) ="";
    my ($group) ="";
    my ($continue_program) = "TRUE";
    my ($error_code) = 0;

   debug_message ("hosts:\n");
    foreach $host (@HOSTS)
    {
        debug_message ("\t$host\n");
    }

   debug_message ("groups:\n");
    foreach $group (@GROUPS)
    {
        debug_message ("\t$group\n");
    }

   # What about the GUI option........
    if ($LAUNCH_GUI eq $TRUE)
    {
        display_message ("Launching GUI...\n");

       run_distributed_command_gui();

       exit_program ("", 0 );
    }

   debug_message ("User prompt setting is: $PROMPT_USER\n");

   if ($PROMPT_USER eq $TRUE)
    {
        $continue_program = confirm_command_execution
($CMD_SPECIFICATION,
                           \@HOSTS, \@GROUPS);
        if ($continue_program eq $FALSE)
        {
            my ($exit_msg) = "Program exited without executing
command.\n";
            exit_program ($exit_msg, 1 );
        }
    }

   # After all the GUI and PROMPT options have been processed,
    # we are now ready to run the script
    my (@results) = run_distributed_command_line ();
    if (-e "/opt/csm/dcem/bin/dLogMgr")
    {
        generate_log_entries(@results);
    }

   debug_message ("Leave function main_driver().\n");
}

#########################################################################
# This the start of the script.
#
#########################################################################

main_driver ();

0;  # return 0 (no error from this script)

#########################################################################
#----------------- This is the END of the Script -------------
#
#########################################################################
```