CA Spectrum®

Command Line Interface User Guide

Release 9.4

CA Technologies Product References

This guide references CA Spectrum®.

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction to Command Line Interface (CLI)

This section contains the following topics:

Overview

The CA Spectrum Command Line Interface (CLI) is a core CA Spectrum component and is installed with the core CA Spectrum product.

You can access CA Spectrum data and can execute CA Spectrum operations from the OneClick user interface. However, if you prefer to execute CA Spectrum operations from the command line, you can use the CLI. For those tasks that you cannot execute in OneClick, CLI is the only CA Spectrum resource available to you.

CLI is a powerful tool, but it does not provide the safeguards that OneClick does, especially related to modeling. CLI must be used by CA Spectrum administrators who understand the potentially harmful effects of haphazardly creating and destroying models and modifying model attributes on a network modeling scheme.

CLI is a flexible option. You can open a CLI session and issue commands from any of the command prompts that are available on your system, such as UNIX, DOS, and Bash.

CLI Commands

CLI commands are similar to UNIX commands, and they can be used with UNIX or DOS commands, especially grep (find), pipes, and redirect symbols. Some CLI commands, however, can conflict with UNIX commands of the same name. For example, the CLI update command can conflict with the UNIX update command.

To avoid conflicts, use **./update** from within the vnmsh directory. When using a script, use the full pathname for the CLI command, for example, <*$SPECROOT*>**/vnmsh/update**.

Note: The CLI update command always provides a response, either a confirmation that the update was successful or a message that the update failed. If you receive no response from CLI when using the update command, type "**which update**". The system likely responds with:

```
/etc/update
```

More information:

Command Descriptions (see page 29)

CLI in Shell Scripts

CLI commands can be incorporated into shell scripts or menu systems to give you a more powerful and versatile method of accessing CA Spectrum data.

Each CLI command sends output reporting the success or failure of the command to standard error. Normal output that is expected as the result of the success of a command, however, is sent to standard output. Each of the commands also generates a return code of zero on success and a non-zero error code on failure. Return codes enable shell scripts using the CLI commands to proceed according to the success or failure of each command.

CLI Components

CLI components are described in this guide as follows:

- Executable commands
- Four environment variables
- The daemon that maintains communication with a SpectroSERVER
- The set of sample shell scripts that incorporate CLI commands

More information:

Command Descriptions (see page 29)
CLI Environment Variables (see page 11)
The CLI Local Server (see page 14)
Sample Scripts (see page 67)

CLI Environment Variables

You can set the following four environment variables for CLI:

CLIMNAMEWIDTH

Displays a model name. By default, the create, seek, and show commands each display a maximum of 16 characters for a model name. However, with the environment variable CLIMNAMEWIDTH, you can specify up to 1024 characters to display for model names. For example, using the C shell:

```
setenv CLIMNAMEWIDTH 32
```

You can set this variable in your .login file, in a script, or simply before you issue a command. You can set or change any number of times in a CLI session, depending on the length of the model names.

CLISESSID

Represents the ID for use in scripts. Set the CLISESSID variable to <$$>, which represents the process ID of the running shell script. This variable is necessary when using cron to run CLI scripts concurrently. For example, using the bash shell:

```
CLISESSID=<$$>; export CLISESSID
```

Also, setting the CLISESSID environment variable to a unique value for each CLI session is required if you run CLI on Windows using the bash shell instead of DOS. You can give a unique timestamp to each bash shell, for example:

```
export CLISESSID='date +%s'
```

SPECROOT

Displays the alarm or event description. The SPECROOT environment variable is required for the show alarms or show events commands when the -x option is specified. This variable gets the description of the alarm or event from the SG-Support directory tree if it can be located so that the output from the commands is expanded.

On UNIX, you can specify the SPECROOT variable in your login shell and can set this variable to the CA Spectrum home directory. For example:

```
SPECROOT=/home/CA Spectrum; export SPECROOT
```

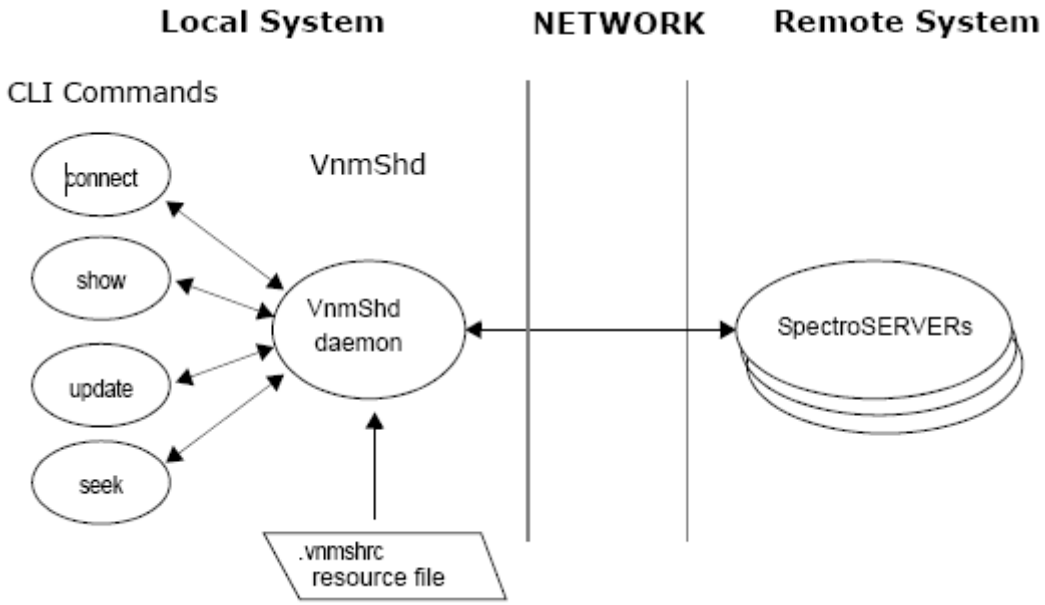
On Windows, see your system documentation for details about setting environment variables.

CLIPATH

Displays the path of the <*$SPECROOT*>/vnmsh directory and the scripts require to use CLI commands.

CLI Architecture

The following image depicts the CLI architecture:



The CLI Local Server, which uses .vnmshrc at startup, performs the following major functions:

- Maintaining a constant network connection with SpectroSERVER. The CLI Local Server prevents disconnection when a command is executed each time. This server maintains a *single* connection to SpectroSERVER regardless of the number of CLI users that are connected to the daemon. The socket connects and disconnects are expensive as far as time and resource usage are concerned.
- Maintaining state information for each CLI user. The 'current' and setjump commands, for example, require the CLI Local Server to store state information. The current command stores a model handle and a landscape handle for use in future commands. The setjump command stores a text string to identify the current position of users in a CA Spectrum landscape.

The Startup File

The .vnmshrc file, the CLI Local Server startup file, is located in the <*$SPECROOT*>/vnmsh directory. This file contains several parameters that control how vnmsh communicates with the SpectroSERVER. These parameters are described in the following list:

vnm_hostname

Specifies the host name of the SpectroSERVER to connect to.

client_handshake_timeout

Specifies the number of milliseconds the client waits for server ID information, when setting up a connection.

Default: 900

server_handshake_timeout

Specifies the number of milliseconds the server waits for client ID information, when setting up a connection.

Default: 900

connect_time_limit

Specifies the maximum number of milliseconds to wait for a connection to the SpectroSERVER.

Default: 1000

listen_backlog

Specifies the number of client requests to the SpectroSERVER held in queue while waiting for prior ones to complete.

Default: 10

vnm_tcp_port

Specifies the TCP port that the vnmsh is using to communicate with the SpectroSERVER when the vnmsh is a SpectroSERVER client.

vsh_tcp_port

Specifies the TCP port where the vnmsh listens for TCP messages when the vnmsh is acting as a server to the client requests such as show, update.

debug_file

Specifies the file to which CLI writes error messages.

max_show_event_length

Specifies the maximum number of characters that are shown when the **show events -x** command is used to display an event message.

Default: 512

The CLI Local Server

The first user to issue the connect command automatically starts the CLI Local Server (VnmShd daemon) on that workstation and establishes a connection to a SpectroSERVER. Only one CLI Local Server per workstation can be running, and that daemon makes only one connection to a SpectroSERVER.

After the CLI Local Server has been started on a workstation, all subsequent users who connect to CLI on that workstation use the same CLI Local Server.

More information:

CLI Environment Variables (see page 11)

Error Checking

CA Spectrum enforces certain rules when you perform tasks in OneClick. For example, rules control the allowable actions when you create or move device models in the different views.

CLI does not enforce these rules and cannot perform any error checking. As a result, CLI lets users create models and place them wherever they want without performing error checking. You see an error if you attempt to use a CLI command in a manner that does not conform to its format.

Chapter 2: Working with Command Line Interface

This section contains the following topics:

Start a CLI Session on UNIX

On a UNIX platform, you can start a CLI session from the shell prompt.

Note: You can use a script to pack up CLI so that it can be sent to another server. For more information, see the *CA Spectrum Distributed SpectroSERVER Administrator Guide*.

Follow these steps:

1. Start the SpectroSERVER to which you want to connect.
2. Navigate to the vnmsh directory in the CA Spectrum installation directory:

   ```
   $ cd <$SPECROOT>/vnmsh
   ```

3. Open the connection:

   ```
   $ connect
   ```

   You are connected to the CLI session.

Start a CLI Session on Windows using DOS Prompt

On the Windows platform, you can start a CLI session from the DOS prompt.

Note: You can use a script to pack up CLI so that it can be sent to another server. For more information, see the *CA Spectrum Distributed SpectroSERVER Administrator Guide*.

For all instances of UNIX (as opposed to CLI) commands in this guide, substitute the equivalent DOS command when necessary. For example, use *find* instead of *grep*.

Follow these steps:

1. For DOS prompt, select Start, Programs, Command Prompt.

   The DOS prompt appears, ready to accept CLI commands.
2. Start the SpectroSERVER to which you want to connect.
3. Navigate to the vnmsh directory in the CA Spectrum installation directory:

   ```
   $ cd <$SPECR00T>/vnmsh
   ```
4. Open the connection:

   ```
   $ connect
   ```

   You are connected to the CLI session.

More information:

UNIX to DOS Conversion (see page 91)

Start a CLI session on Windows using Bash Prompt

On the Windows platform, you can also start a CLI session from a bash shell prompt.

Note: You can use a script to pack up the CLI program so that it can be sent to another server. For more information, see the *CA Spectrum Distributed SpectroSERVER Administrator Guide*.

Follow these steps:

1. Click Start, Programs, and Command Prompt.

   The DOS prompt appears.
2. From the DOS prompt, type **bash**.
3. Click Start, Run, and type **bash -login**.

   You can start a CLI session from a bash shell prompt.
4. Start the SpectroSERVER to which you want to connect.

5. Navigate to the vnmsh directory in the CA Spectrum installation directory:

   ```
   $ cd <$SPECROOT>/vnmsh
   ```

6. Open the connection:

   ```
   $ connect
   ```

   You are connected to the CLI session.

Example Usage

The following examples demonstrate how to use CLI commands for common tasks in CA Spectrum.

Create a User Model

A User model gives a user access to CA Spectrum. Users are identified by login IDs.

Note: Before you start a CLI session, verify that the User Model is created and the SpectroSERVER to connect is started.

Follow these steps:

1. Connect to the SpectroSERVER.

   ```
   $ cd <$SPECROOT>/vnmsh
   $ ./connect
   ```

   You are connected to the SpectroSERVER.

   Note: If you have trouble connecting, verify for error messages. For more information, see Error Messages (see page 71).

2. Determine the model type handle for the model type you want to create using the show command. In this case, it is a model of type User. Enter this command:

   ```
   $ ./show types | grep User
   ```

   Note: The "./" is important. Some UNIX systems use the show command for reading email. If the "." is not the first path in the users environment, then "./" are required.

   A list of model types that include the string 'User' appears with the User model type listed first.

   ```
   Handle     Name          Flags
   0x10004    User          V,I,D
   0x1040a    UserGroup     V,I,D
   0x1040f    DefUserGroup  V,I,N,U,R
   0xaa000d   GenSwUserPort V,I,D
   0xf000d    ForeUserAgen  V,I,D
   0xaf000c   ForeUserApp   V,I,D
   ```

3. List the attributes using the show command for the User model type and determine the attribute ID for the model name attribute. You need this attribute ID to create the model. Enter this command:

   ```
   $ ./show attributes mth=0x10004 | grep -i name
   ```

   A list of User model type attributes including the model name attribute appears.

   ```
   Id        Name                 Type          Flags
   0x10000   Modeltype_Name       Text String   R,S,M
   0x1006e   Model_Name           Text String   R,W,G,O,M,D
   0x10074   User_Full_Name       Text String   R,W,O,D
   0x1155f   gib_mtype_nameText   String        R,W,S,D
   0x11560   gib_mtype_name_menu  Text String   R,W,S,D
   0x11561   gib_model_name       Text String   R,W,D
   0x11563   gib_model_name_menu  Text String   R,W,D
   0x1197d   WatchNames           Tagged Octet  R,W,D
   ```

4. Create the model using the create command with model type handle, the attribute ID for the model name, and the value (the login ID name) for the user. In this example, the user login ID is j_doe. Enter this command:

   ```
   $ ./create model mth=0x10004 attr=0x1006e,val=j_doe
   ```

   A system message resembling the following command confirms that the model is created:

   ```
   created model handle = 0xbe0001b
   ```

   Note: All handles and IDs used in these examples are fictitious. The model handle for the model that you created is different; model handles are created by the system.

More information:

Error Messages (see page 71)

Modify a Model Attribute

This section provides an example to change the value of a model attribute using CLI commands. In particular, this example demonstrates how to change the community string attribute value for the model (j_doe) created in a User Model. For more information, see Create a User Model (see page 17).

Follow these steps:

1. Determine the j_doe model handle, and then set j_doe as the current model:

   a. $ ./show models | grep j_doe

      The following information about the j_doe model appears:

      ```
      0xbe0001b        j_doe(Active)        0x10004        User
      ```

   b. $ ./current mh=0xbe0001b

      The system confirms that j_doe is the current model:

      ```
      current model is 0xbe0001b
      current landscape is 0xbe00000
      ```

2. Determine the ID for the community string attribute.

   Note: For the sake of brevity, this step shows a known portion (community string) of the attribute name as an argument to the grep command. If you do not know the name of the attribute, you can show and scan all attributes for the model to determine the correct attribute name and its attribute ID.

3. Enter this command:

   ```
   $ ./show attributes | grep -i community_string
   ```

   The attribute ID, the attribute name, and the community string value appear:

   ```
   0x1007a        User_Community_String        ADMIN,0
   ```

   CA Spectrum assigns a default value of ADMIN,0 to all user models when they are created. ADMIN,0 confers full administrative privileges in CA Spectrum to user models.

4. Change the administrative permission level from ADMIN,0 to example permission level ADMIN,5 (read-only) for the j_doe model using the update command. Enter this command:

   ```
   $ ./update attr=0x1007a,val=Subnet3,5
   ```

   An entry showing the change in the attribute value is returned:

   ```
   Id        Name                  Iid    Value
   0x1007a   User_Community_String        ADMIN,5
   ```

   The Iid attribute has no value here because it applies only to list attributes. For more information, see the *CA Spectrum Administrator Guide*.

Create and Modify a Model in One Step

This section provides an example of how to create a model and replace a default attribute value with another value in a single command string. You can only execute a complex command of the type shown in this section if you know the values of the relevant model identifiers that are provided for the command before you attempt to execute.

The following example uses the parameter values introduced in Creating a User Model and Modifying a Model Attribute:

```
$ ./create model mth=0x10004 attr=0x1006e,val=j_doe attr=0x1007a,val=ADMIN,5
```

More information:

Create a User Model (see page 17)
Modify a Model Attribute (see page 18)

Sample CLI Script File–Create a New User

You can execute shell scripts that incorporate CLI commands from the bash prompt in the Windows platform simply as you execute the command from the shell command prompt on UNIX.

This following example demonstrates how a script can be used to create a CA Spectrum user model.

```
#
#  Check to see if CLIPATH is set.  If it is not then we will have to create it.
#
#  Setup a variable to point to the /install_area/vnmsh directory so we can
#  find the commands we need.
#
if [ -z "$CLIPATH" ]
then
     CLIPATH=/usr/data/spectrum/7.0/vnmsh
     export CLIPATH
fi
```

```
#
#  Test to make sure the CLIPATH points to a valid directory
#
if [ ! -d $CLIPATH ]
then
        echo "ERROR: could not find $CLIPATH"
        echo "Please find the correct path to the vnmsh directory and set"
        echo "the CLIPATH environment variable to it."
        exit 0
fi
#
#  Now check to see how many command line arguments there are.  If there are
#  none, then echo a usage message.  If there is one, that is all we really
#  need to create a new user...  If there is a second argument then we can
#  set the Community_String at the same time.
#
#  This setup is only for creating a user on the local system or what the
#  .vnmshrc file points to for the vnm_hostname.  A third field could be
#  added that accepts the vnm_hostname to connect to.
#
#  Optionally, the getopts shell command can be used to parse "switches" to
#  the script:  -n for name, -c for community string and -v for vnm_hostname.
#
#  (NOTE:  getopts should be located in /usr/bin/getopts if the script is
#   done in bourne shell (sh).  k-shell has a built in getopts function)
#
if [ $# -eq 0 ]
then
        echo "Usage:  $0 username [Community_String]"
        exit 1
elif [ $# -eq 1 ]
then
        command="attr=0x1006e,val=$1"
        flag=0
elif [ $# -eq 2 ]
then
        command="attr=0x1006e,val=$1 attr=0x1007a,val=$2"
        flag=1
fi

#
#  Okay, we should be all set now to go ahead and create the new user.
#  The first thing we have to do is connect.
#
$CLIPATH/connect
```

```
#
#  Now let's check the exit status of the connection to see if we got in...
#
if [ $? -ne 0 ]
then
        echo "ERROR:  could not connect to SpectroSERVER.  $0 exiting"
        exit 0
fi
#
#  Okay if we made it this far then we have a connection.  Let's try the
#  create command.
#
$CLIPATH/create model mth=0x10004 $command
#
#  Now we check the exit status again and see if we actually created a model.
#
if [ $? -ne 0 ]
then
        echo "ERROR:  could not create a new user.  $0 exiting"
        exit 0

else
        echo -n "New user $1 created"
        if [ $flag -eq 1 ]
then
        echo " Community_String was set to $2"
fi
        echo "Successfully created new model... exiting."
fi
$CLIPATH/disconnect
exit 1
```

Event Report Generation

The CLI keeps a list of the 2000 most current events that occur on a landscape. However, if many events occur on a landscape, the most recent events are approximately one hour old.

You can set the SPECROOT environment variable when using the -x option with the show events command. The following command is an example of to run event reports using CLI:

```
$ ./show events | more
$ SPECROOT=/home/spectrum; export SPECROOT
$ ./show events -x > event_rpt
```

Model Switch

The jump and setjump commands are useful in scripts where you can move back and forth between different models. The setjump command lets you assign a text string to represent a model handle and its corresponding landscape handle. Then you can use the jump command with that text string to retrieve that information as the current model handle. For example:

- $ ./current mh=0xb6000f8

  ```
  current model is 0xb6000f8
  current landscape is 0xb600000
  ```

- $ ./setjump emme

  ```
  model 0xb6000f8 and landscape 0xb600000 stored under emme
  ```

- $ ./jump emme

  ```
  current model is 0xb6000f8
  current landscape is 0xb600000
  ```

Create a Troubleshooter Model

You can create and associate Troubleshooter models with User models using the CLI. Once created and associated, these troubleshooters can be assigned alarms and receive email notification that they must investigate and resolve the alarms.

The following procedure describes how to create a TroubleShooter model and associate it with a User model. The User model that is created in Create a User Model (see page 17) is used as an example.

Follow these steps:

1. Navigate to the *<$SPECROOT>*/vnmsh directory.
2. Connect to the SpectroSERVER by typing the following command at a command prompt (example from a bash shell with a $ prompt):

   ```
   $ ./connect
   ```

3. Determine the TroubleShooter model type. Enter this command:

   ```
   $ ./show types | grep -i trouble
   ```

   The TroubleShooter model type entry is returned.

   ```
   0x10372        TroubleShooter        V,I
   ```

4. Determine the TroubleShooter model type EmailAddress attribute ID. Enter this command:

   ```
   $ ./show attributes mth=0x10372 | grep -i email
   ```

   The EmailAddress entry appears:

   ```
   0x11d24        EmailAddress         TextString    R,W,D
   ```

5. Create a TroubleShooter model using the CLI create command:

   ```
   $ ./create model mth=0x10372
   attr=0x1006e,val=j_doe_fixit
   attr=0x11d24,val=j_doe@aprisma.com
   ```

   A system message resembling the following command confirms that the model is created:

   ```
   $ created model handle = 0xbe0001c
   ```

   Note: All handles and IDs used in examples are fictitious. The model handle for the model you created are different. It is whatever your system creates for it.

6. Create the association between the j_doe User model (mh=0xbe0001b) and the j_doe_fixit TroubleShooter model (mh=0xbe0001c) using the CLI create command:
   ```
   $ ./create association rel=Is_Assigned lmh=0xbe0001b rmh=0xbe0001c
   ```

   A system message similar to the following command confirms that the association was created:
   ```
   $ create association successful
   ```

Assign an Alarm to a Troubleshooter

This section describes how to list alarms and assign an alarm to a troubleshooter using CLI commands.

Follow these steps:

1. List alarms using the show command.

   This step shows how to find only those alarms with an alarm_severity of MAJOR.

   ```
   $ ./show alarms | grep MAJOR
   ```

   A list of MAJOR alarms appears. For example:

```
7509   09/27/2000   14:46:44   0xd80008   0xa6000df   duncan    9E133_36   MAJOR   No
7645   09/27/2000   14:47:16   0xd80008   0xa60025e   infinity  9H422_12   MAJOR   No
7518   09/27/2000   14:47:01   0xd80008   0xa6000eb   rugone    9E132_15   MAJOR   No
7979   09/27/2000   14:53:12   0xf40002   0xa600161   FDDI2     FddiMAC    MAJOR   No
8018   09/27/2000   14:53:13   0xf40002   0xa6003da   FDDI FNB  FddiMAC    MAJOR   No
7512   09/27/2000   14:46:47   0xd80008   0xa6000af   ruthere   9A426_02   MAJOR   No
```

2. Select an alarm to which you want to assign a troubleshooter. In this example, alarm ID 7512 for the 9A426-02 device is selected.

3. Select a troubleshooter to assign to the alarm. In this example, the j_doe_fixit TroubleShooter model that was created in Create a Troubleshooter Model (see page 23) is selected.

   Note: Use the TroubleShooter model handle, 0xa600722, rather than the TroubleShooter model name, j_doe_fixit, to specify the troubleshooter in the update command in the next step.

4. Assign the alarm to the troubleshooter using the update command:

   ```
   $./update alarm aid=7512 assign=0xa600722
   ```

   A system message similar to the following example confirms that the troubleshooter was assigned to the alarm:

   ```
   $ update: successful
   ```

   The person who is represented by the j_doe_fixit model has now been assigned to the alarm. This person receives an email notification of the alarm assignment.

Create a Global Collection

You can create a global collection and can set the search criteria in a CLI session using GUID. A unique identifier (GUID) is a key attribute to create global collection. The GUID is required for global collection to function properly. You can obtain a GUID through an action to the VNM model.

Note: A global collection without a GUID in the CLI is invalid. When a global collection is created using OneClick, a GUID is automatically created. For more information, see the *CA Spectrum Modeling and Managing Your IT Infrastructure Administrator Guide*.

Follow these steps:

1. Enter the following command:

   ```
   update action=(Action to get a unique identifier) 0x10474 mh= (VNM Model Handle)
   ```

   A GUID is created.

   Note: The action to get a new GUID is 0x10474, which is the same as the global collection model type.

2. Enter the following command to create the global collection using GUID:

   ```
   create model mth=(Model Type for global collection) 0x10474 attr=(GUID) 0x12e56,
   val=(Previous value you received) 4a85b9af-0d52-1000-017f-0013727f8c0a
   ```

   The Global Collection is created and appears in the Navigation pane under global collections.

Example: Create a global collection with or without XML string

Enter the following command to create a global collection with or without XML string:

```
update action=(Action to get GUID)0x10474 mh=(Landscape)

Printed:
update action: successful
Response has 1 attributes:
 0) Attribute 0x0 text: EXAMPLE GUID(4a85b9af-0d52-1000-017f-0013727f8c0a)

create model mth=(Model Type for global collection)0x10474
attr=(GUID)0x12e56,val=(Previous value you
received)4a85b9af-0d52-1000-017f-0013727f8c0a
attr=(dynamicCriteriaXML)0x12a6a,val=(XMLString)'<search-criteria><filtered
models><equals-ignore-case><model-name>sometext</model-name></equals-ignore-c
ase></filtered-models></search-criteria>'
Printed:
created model handle = New model handle(0x78101069)
```

Note: You can specify the dynamicCriteriaXML (0x12a6a) attribute with the create command or you can update the model later.

More information:

Command Descriptions (see page 29)

Suppress Headers in CLI Output

To suppress the headers in CLI output, you can create a file that includes the functions provided in the following section and can refer this file at the top of each script.

The functions in the following procedure call CLI commands and strip the header information from the output of the commands.

Follow these steps:

1. Create a file named StripHeaders in your scripts directory.
2. Include the following functions in the StripHeaders file:

```
tcreate() # only needed for the createalarm
    { # and create event commands
    $CLIPATH/create $@ | tail +2
    }
tseek()
    {
      $CLIPATH/seek $@ | tail +2
    }
tshow()
    {
    $CLIPATH/show $@ | tail +2
    }
tupdate()
    {
    $CLIPATH/update $@ | tail +2
    }
```

3. Include the name StripHeaders at the top of your CLI script as follows:
   `. StripHeaders`
4. Call the tcreate(), tseek(), tshow(), and tupdate() functions instead of the corresponding CLI command whenever you want to strip the headers from CLI output.

   For example, the following line generates the output of the show models command without the CLI header information:
   `tshow models`

Chapter 3: Command Descriptions

This chapter provides descriptions of CLI commands and output.

This section contains the following topics:

Command Descriptions Overview

You can make changes to the CA Spectrum knowledge base without the safeguards that are available in CA Spectrum using the CLI. A system crash or database corruption can result if you specify incorrect information. Therefore, proceed with caution when you use the create, destroy, or update commands.

Note: Use the CLI command parameters for creating and managing response time tests with CLI.

For more information, see the *CA Spectrum Service Performance Manager User Guide*.

ack alarm–Acknowledges Alarm

The ack alarm command acknowledges the alarm specified by alarm_id in the landscape that is specified by landscape_handle. If landscape_handle is not specified, the command acknowledges the alarm that is specified by alarm_id in the current landscape.

Acknowledging one alarm for a model means that you acknowledge only that alarm and no other alarm for that model.

The command has the following format:

```
ack alarm aid=<alarm_id> [lh=<landscape_handle>]
```

If ack alarm is entered with a valid alarm_id and a valid landscape_handle, the following message is displayed:

```
ack alarm: successful
```

Example: ack alarm

```
$ ack alarm aid=42 lh=0x400000
ack alarm: successful
```

connect–Connects to SpectroSERVER

The *connect* command connects the user of the CA Spectrum Command Line Interface to the SpectroSERVER running on host system, *hostname.* This command also sets the landscape that is specified by landscape_handle to be the current landscape. If the CLI Local Server is not already running, the connect command starts it.

The command has the following format:

```
connect [<hostname>] [lh=<landscape_handle>][vnmsocket=<vnmsocket>]
```

hostname

(Optional) If hostname is not specified, the command connects the user to the host specified in the CLI resource file .vnmshrc.

Note: CA Spectrum Command Line Interface does not support localhost or the 127.0.0.1 option. To connect to localhost, you can specify the actual hostname or do not specify any parameter.

landscape_handle

(Optional) If landscape_handle is not specified, the command sets the current landscape to the landscape of the host name specified.

vnmsocket

(Optional) If vnmsocket is not specified, the command connects to the SpectroSERVER using the socket specified in the .vnmshrc file. You can use vnmsocket to connect to another SpectroSERVER on a different port connection that is defined by vnmsocket.

On Unix, error messages that are reported by the CLI Local Server are displayed in the console window. On Windows, these errors are displayed in the user bash shell window.

Example: connect

```
#! /usr/bin/sh
# A sample script to get alarms of a specific
# severity and set the CLISESSID

if [$# !=1]
then
    echo “Usage: $0 <alarm severity>”
    exit 0
fi

CLISESSID=$$

$SPECROOT/vnmsh/connect
$SPECROOT/vnmsh/show alarms | grep -i $1
$SPECROOT/vnmsh/disconnect

exit 0
```

If the command is successful, the following message is displayed:

```
connect: successful hostname
current landscape is <landscape_handle>
```

Hostname is the user-entered SpectroSERVER host or the host that is specified in the .vnmshrc file. landscape_handle is the user-entered landscape or the landscape for the host.

More information:

CLI Environment Variables (see page 11)
The Startup File (see page 13)

Considerations when Using connect Command

The following are important considerations when using the connect command:

- The user on a terminal must use the connect command to initiate communications. The same user must use the disconnect command to terminate communication with the SpectroSERVER.
- Once the first user has entered the connect command, the CLI Local Server is connected to the SpectroSERVER.
- Other CLI users using the same CLI Local Server can connect only to those SpectroSERVERs that are in the landscape map of the initial SpectroSERVER. Once all users have disconnected, use the connect command to connect to SpectroSERVERs in a different landscape map.
- To successfully connect to the SpectroSERVER, the first user of the connect command must be defined as a user in the CA Spectrum database of the original SpectroSERVER.
- Windows users running CLI in the bash shell must also define CLISESSID.
- The terminal device for a particular user is determined using the ttyslot(3V) function.
- Cron scripts are not attached to a ttyslot. As a result, the ttyslot function returns 0 for all cron scripts. That is, two CLI scripts running as cron scripts at the same time appear as one CLI user to the CLI Local Server, which leads to unpredictable results. Therefore, you must insert a line at the top of the script to export the environment variable CLISESSID. Set CLISESSID to a unique numeric value. CLI can now distinguish between the different cron scripts.

  The following example defines a unique CLI session ID within a script:

  ```
  CLISESSID=$$; export CLISESSID
  ```

- This example sets the CLISESSID as the process ID of the shell running the script. CLI uses CLISESSID to identify a user when the ttyslot function returns zero. Set CLISESSID once for each CLI session. If a CLI script that is running as a cron script calls other CLI scripts, only the top-level script will set the CLISESSID environment variable. The other CLI scripts run under the same process ID unless you invoke a new shell (#!/bin/sh) at the top of the script. To invoke a new shell in the other scripts, export the CLISESSID and then connect and disconnect again.
- In some environments or configurations, even when the command is entered from a command line, the ttyslot function may return zero. In such a case, the connect command returns the following error:

  ```
  connect: variable CLISESSID not set
  ```

  In this situation, set CLISESSID either from the command line or from .cshrc or other startup file.

- The CLI uses the user name and terminal device to identify each CLI user. A user running multiple scripts from a terminal device at one time will appear to CLI as same user. The CLI can give unpredictable results if a script is running in the background and another script is running in the foreground, or multiple scripts are running in the background.

  For example, Script A1 sets the current model to be Model A. Script B1, which is run by the same user from the same terminal device, sets the current model to be Model B. If Script A1 performs an update command on Model A, the update command is also performed on Model B of Script B1.

  Run only one CLI session from a particular terminal device at one time. To run multiple CLI sessions at once, run them from separate terminal device, or run them using the at(1) or batch(1) commands with the CLISESSID environment variable set to a unique value for each.

create–Create Object

Use the create command to create an object.

Note: For information about how to create a model in a secure domain, run ./create to display a usage statement.

The command has the following format:

```
create  model ip=<IP Address | Low_IP-High_IP>
[sec_dom=Secure_Domain_Address][comm=Community_Name] [to=Time_Out] [tc=Try_Count]
[lh=landscape_handle] |
create  model mth=model_type_handle [attr=attribute_id,val=value ...]
[lh=landscape_handle] |
create  association rel=relation lmh=left_model_handle rmh=right_model_handle
create  alarm [-nr] sev=alarm_severity cause=probable_cause_id [mh=model_handle] |
create  event type=event_type text=event_text [mh=model_handle|lh=landscape_handle]
```

More information:

CLI Environment Variables (see page 11)

create alarm

The command create alarm creates an alarm with severity alarm_severity and cause probable_cause_id for the model with model_handle. Valid alarm severity options are: CRITICAL, MAJOR, MINOR, OK, MAINTENANCE, SUPPRESSED, or INITIAL. By default, the new alarm replaces an existing alarm.

If create alarm is entered with a valid alarm_severity, a valid probable_cause_id, and a valid model_handle, the created entry in the alarm table is displayed. The create time is displayed in hh:mm:ss format.

Example: create

```
$ create alarm sev=CRITICAL cause=0x10308 mh=0x400134

ID     Date          Time        PCauseID    MHandle     MName    MTypeName     Severity    Ack
984    05/11/2000    12:33:27    0x10308     0x400134    12.84    Bdg_CSI_CN    CRITICAL    No
```

create association

The command create association, creates an instance of the relation (an association) between the model with left_model_handle and the model with right_model_handle.

If create association is entered with a valid relation between a valid left_model_handle and right_model_handle, the following message is displayed:

```
create association: successful
```

Example: create association

```
$ create association rel=Collects lmh=0x400009 rmh=0x400134
create association: successful
```

create event

The create event command, creates an event with type event_type and text event_text for the model that is specified by model_handle. If a landscape_handle is specified, the event is created for the user model that created the event.

Note: In previous versions of the CLI, the event was created for the landscape model.

If model_handle or landscape_handle is not specified, the event is created for the user model that created the event. Or the the event is created for the current model if one has been specified. Some events in CA Spectrum lack an associated model. For example, when an application connects to the SpectroSERVER, no model is associated with the event.

If create event is entered with a valid event_type, valid event_text, and a valid model_handle or landscape_handle (if present), the entry appears in the event table. The create time is displayed in hh:mm:ss format.

The event_type command (also named an event code in CA Spectrum) is a 4-byte hexadecimal number. The two most significant bytes specify the developer ID for the event (0001 for CA Spectrum-generated event codes), and the two least significant bytes are a unique event identifier. Not all event types include user-entered text. Examples of such event types are those that include the variable {S 0} in their event format files. For those event types that do not include user-entered text, the event_text parameter is ignored but must still be present on the command line.

For more information, see the *CA Spectrum Certification User Guide*.

Example: create event

```
$ create event type=0x1061a text="fan down" mh=0x40013

Date          Time        Type       MHandle     MName    MTypeName
05/11/2000    12:39:42    0x1061a    0x400134    12.84    Bdg_CSI_CNB20
```

create model

You can specify create model with an IP address or with a model type handle. In either case, the system creates the model in the landscape that is specified by landscape_handle. If landscape_handle is not specified, the command creates the model in the current landscape.

Note: The model_name attribute is required only when creating a User model.

- If you specify create model with an IP address, the system finds the object at the specified ip_address and creates a model for it. The model has all of the properties of that object including any associated children. For example, if the object is a hub, the create model command creates a model of a hub with all of its ports.
- You can specify an IPv4 address or an IPv6 address. IPv6 ranges are not supported and this command does not support the setting of attribute IDs.
- To create several models at once, you can define a range of IP addresses with the create model command. Specify the Low_IP and High_IP parameters, separated by "-". If the Community_Name is not specified, the newly created model is of type "Pingable". If the Community_Name is specified, the device is modeled to the appropriate model type. The Try_Count and Time_Out options are similar to options in the OneClick 'Create Model by IP' dialog.
- If you specify the create model command with a model type handle, the system creates a model of type model_type_handle. You can then set the value of one or more attributes for the created model.
- When you specify the create model command with a model type handle, you can also specify multiple attributes in that one command. Specify multiple 'attribute_id, value' pairs, separating each pair from adjacent pairs by a space.

- The attribute values that the user specifies when creating a model of a particular model type while using OneClick should be specified in the create model command. Otherwise, Inference Handler errors can occur within a SpectroSERVER when the model is created. For example, when creating a Hub_CSI_IRM3 model using OneClick, a window is displayed in which you can enter values for Model Name, Network Address, Community String. Specify values for these attributes when using the create model command to create a model of the same type using the CLI.
- If create model is specified with a valid model_type_handle and valid attribute_id, value pairs (if present), the created model handle is displayed.
- If create model is specified with a valid ip_address, the created model handle is displayed.

Example: create model

```
$ create model mth=0x102d attr=0x12d7f,val=132.177.12.84
attr=0x1006e,val=12.84lh=0x400000
created model handle = 0x400134

$ create model ip=206.61.231.1-206.61.231.5

Creating model for IP=206.61.231.1
created model handle =  0x9a00259
Creating model for IP=206.61.231.2
create model: DCM device unreachable
Creating model for IP=206.61.231.3
create model: DCM device unreachable
Creating model for IP=206.61.231.4
create model: DCM device unreachable
Creating model for IP=206.61.231.5
created model handle =  0x9a0025a
```

Note: By default, the create command displays a maximum of 16 characters for the model name. However, with the environment variable, CLIMNAMEWIDTH, you can specify a different number of characters (up to 1024) to be displayed for model names.

current–Sets Model or Landscape

The current command sets the model that is specified by model_handle to be the current model. Or this sommand sets the landscape that is specified by landscape_handle to be the current landscape. If the model_handle and the landscape_handle are not specified, current displays the current model handle and the current landscape handle.

When the user sets a current model, the CLI sets the current landscape to the landscape that contains the model. When a user sets the current landscape, the CLI sets the current model as undefined.

Separate current model and current landscape values are maintained for each session that is connected to the CLI Local Server.

The current command retains state information, the current model and the current landscape, for example, only for the session that named it.

This command has the following format:

```
current [mh=<model_handle>|lh=<landscape_handle>]
```

- If a valid model_handle is specified as input, the following message is displayed:

  ```
  current model is <model_handle>
  current landscape is <current_landscape_handle>
  ```

- If a valid landscape_handle is specified as input, the following message is displayed:

  ```
  current model is undefined
  current landscape is <landscape_handle>
  ```

- If model_handle and landscape_handle are not specified, the following message is displayed:

  ```
  current model is <current_model_handle>
  current landscape is <current_landscape_handle>
  ```

- If model_handle and landscape_handle are not specified and current model is not defined, the following message is displayed:

  ```
  current model is undefined
  current landscape is <current_landscape_handle>
  ```

Examples: current

```
$ current mh=0x400142
current model is 0x400142
current landscape is 0x400000

$ current lh=0x500000
current model is undefined
current landscape is 0x500000

$ current
current model is undefined
current landscape is 0x500000
```

Note: The current landscape always contains a value because it is set by the connect command.

destroy–Destroys Object

Use the destroy command to destroy an object. This command has the following format:

```
destroy model [-n] mh=model_handle |
destroy association [-n] rel=relation lmh=left_model_handle rmh=right_model_handle|
destroy alarm [-n] aid=alarm_id [lh=landscape_handle]
```

-n

If the -n (no prompt) option is specified with the destroy command, the system does not prompt for confirmation. This option is useful in CLI scripts.

Unless the -n option is specified, one of the following messages is always displayed:

```
destroy model: are you sure?
destroy association: are you sure?
destroy alarm: are you sure?
```

Valid responses are y, yes, Y, Yes, n, no, N, and No.

destroy alarm

Destroys the alarm specified by alarm_id in the landscape that is specified by landscape_handle. Unless the -n option is specified, destroy alarm prompts you for confirmation before destroying the alarm. If the landscape_handle is not specified, the command destroys the alarm that is specified by alarm_id in the current landscape. Use the show alarms command to determine the alarm_ids for a model.

If the destroy alarm command is entered with a valid alarm_id and a valid landscape_handle, the following message is displayed:

```
destroy alarm: successful
```

Examples: destroy alarm

```
$ destroy alarm aid=300
destroy alarm: are you sure? y
destroy alarm: successful
```

destroy association

Destroys the association (instance of the relation) between the model with left_model_handle and the model with right_model_handle. Unless the -n option is specified, destroy association prompts you for confirmation before destroying the association.

If destroy association is entered with a valid relation between a valid left_model_handle and right_model_handle, the following message is displayed:

```
destroy association: successful
```

Example: destroy association

```
$ destroy association rel=Lost_and_Found lmh=0x400001 rmh=0x40h0142
destroy association: are you sure? y
destroy association: successful
```

destroy model

Destroys the model with the specified model_handle. Unless the n option is specified, destroy model prompts you for confirmation before destroying the model.

If destroy model is entered with a valid model_handle, the following message is displayed:

```
destroy model: successful
```

Example: destroy model

```
$ destroy model mh=0xa600715

Following model will be destroyed:
Model_Handle      ->     0xa600715
Model_Type_Handle ->     0x10004
Model_Name        ->     garciaparra
Model_Type_Name   ->     User

destroy model: are you sure? y
destroy model: successful
```

disconnect–Disconnects from SpectroSERVER

Use the disconnect command to disconnect the CLI user from the currently connected SpectroSERVER.

This command has the following format:

```
disconnect
```

If the command is successful, the following message is displayed, where host name is the name of the SpectroSERVER host to which the user was connected:

```
disconnect: successful from <hostname> or <IP address> - connected for xx hours, yy minutes
```

More information:

stopShd–Terminates CLI Local Server (see page 60)

jump–Jumps to Saved Model or Landscape

The jump command jumps to the previously saved model and landscape. The jump command sets the current model and the current landscape to be the model and landscape that were saved under the label text_string by the setjump command. If text_string is not specified, a list of text_strings that were given in previous setjump commands is displayed.

The command has the following format:

```
jump [<text_string>]
```

- If jump is entered with a valid text_string that has been previously defined, the new current model and the current landscape are displayed:

  ```
  current model is <current_model_handle>
  current landscape is <current_landscape_handle>
  ```

- If jump is entered without a text_string, a list of the currently defined text_strings is displayed. For example:

  ```
  text_string1
  text_string2
  --
  ```

- If jump is entered and the new current model is undefined, the following message is displayed:

  ```
  current model is undefined
  current landscape is <current_landscape_handle>
  ```

Example: jump

```
$ jump tutorial
current model is 0x400142
current landscape is 0x400000
```

More information:

setjump–Saves Model and Landscape (see page 43)

seek–Locates a Model

Use the seek command to locate a model. The seek command finds the model(s) in the landscape that is specified by landscape_handle that possess the specified value for the attribute that is specified by attribute_id. If landscape_handle is not specified, the command finds the model(s) in the current landscape that possess a value for the attribute with the specified attribute_id. You can also use a wildcard (*) with seek to find instances of models that contain a specified substring. If you enter a null value, you can find all models that have no name (for example, attr=0x1006e).

You cannot search for a one-character attribute value using the seek command. Attempting such a search returns an error.

The command has the following format:

```
seek [-i] [-s]  attr=attribute_id,val=value [lh=landscape_handle]
```

The options can be used in any order, for example, -i -s, or -s -i.

-i

If the -i (ignore case sensitivity) option is specified with the seek command, then the model information that is specified with the val parameter is returned without regard to case.

-s

If the -s (substrings allowed) option is specified with the seek command, the model information that is specified with the val parameter is returned with substrings, if applicable.

If seek is entered with a valid attribute_id and a valid value, all matching models are displayed in the following format:

```
MHandle      MName   MTypeHnd          MTypeName
modelhandle  name    modeltypehandle   name
```

If no matching models are found, the following message is displayed:

```
seek: no models found
```

Note: By default, the seek command displays a maximum of 16 characters for the model name. However, with the environment variable CLIMNAMEWIDTH, you can specify a different number of characters (up to 1024) to be displayed for model names.

Examples: seek

```
$ seek attr=0x1006e,val=CA Spectrum

MHandle       MName      MTypeHnd   MTypeName
0xb100018     spectrum    0x1004     User
0xb10008d     spectrum    0x820000   ScmConfig
```

```
$ seek attr=0x1006e,val=CA Spectrum

MHandle       MName      MTypeHnd   MTypeName
0xb100018     spectrum    0x820000   ScmConfig

$ seek attr=0x1006e,val=SPE
seek: no models found

$ seek attr=0x1006e,val=spe lh=0xb100000
seek: no models found

$ seek -i attr=0x1006e,val=CA Spectrum

MHandle       MName      MTypeHnd   MTypeName
0xb10018      spectrum    0x10004    User
0xb1008c      spectrum    0x820000   ScmConfig
0xb1008d      spectrum    0x820000   ScmConfig

$ seek -i -s attr=0x1006e,val=CA Spectrum

MHandle       MName      MTypeHnd   MTypeName
0xb10018      spectrum    0x10004    User
0xb10089      spectrum    0x820000   ScmConfig
0xb1008c      spectrum    0x820000   ScmConfig
0xb1008d      spectrum    0x820000   ScmConfig

$ seek -i -s attr=0x1006e,val=CA Spectrum lh=0xb100000

MHandle       MName      MTypeHnd   MTypeName
0xb10018      spectrum    0x10004    User
0xb10089      spectrum    0x820000   ScmConfig
0xb1008c      spectrum    0x820000   ScmConfig
0xb1008d      spectrum    0x820000   ScmConfig

$ seek -s attr=0x1006e,val=CA Spectrum

MHandle       MName      MTypeHnd   MTypeName
0xb10008c     spectrum    0x820000   ScmConfig

$ seek attr=0x110df,val=0.0.C.18

seek: no models found

$ seek -s attr=0x110df,val=0.0.C.18

MHandle       MName        MTypeHnd      MTypeName
0xb100070     frog10       0x210022      Rtr_CiscoIGS
0xb100072     frog10_1     0x220011      Gen_IF_Port
0xb10005b     cisco rtr    0x210022      Rtr_CiscoIGS
0xb100070     frog10_2     0x220011      Gen_IF_Port
0xb100070     cisco rtr_1  0x220011      Gen_IF_Port
0xb100070     cisco rtr_2  0x220011      Gen_IF_Port
```

```
$ seek attr=0x1006e,val=spe*

MHandle       MName           MTypeHnd    MTypeName
0xb10018      spectrum         0x10004     User
0xb10089      spectrum         0x820000    ScmConfig
0xb1008d      spectrum         0x820000    ScmConfig

$ seek attr=0x1006e,val=

MHandle       MName        MTypeHnd      MTypeName
0xd00258      0x102c8                    Physical_Addr
0xd002f8      0x102c8                    Physical_Addr
0xd00368      0x820000                   ScmConfig
0xd00259      0x102c8                    Physical_Addr
0xd002f9      0x102c8                    Physical_Addr
0xd00301      0x102c8                    Physical_Addr

$ seek attr=0x12d7f,val=192.168.93.*

MHandle       MName              MTypeHnd      MTypeName
0x28000190    192.168.93.14      0xd0004       HubCSIEMME
0x28000190    192.168.93.14      0xd0004       HubCSIEMME
0x280001a0    192.168.93.14_Sy   0x23001c      System2_App
0x28000198    192.168.93.14_St   0x590006      RMONApp
0x28000191    192.168.93.14_A    0xd000a       CSIIfPort
0x280001a1    192.168.93.14_IC   0x230012      ICMP_App
0x28000199    192.168.93.14_E    0x590013      RMONEthProbe
0x280001a2    192.168.93.14_UD   0x230019      UDP2_App
0x280001c2    DLM App            0x830001      DLM_Agent
0x2800019a    192.168.93.14_E    0x590013      RMONEthProbe
0x28000192    192.168.93.14_B    0xd000a       CSIIfPort
0x2800019b    192.168.93.14_E    0x590013      RMONEthProbe
```

More information:

CLI Environment Variables (see page 11)
show–Displays Object (see page 45)

setjump–Saves Model and Landscape

The setjump command saves the current model handle and current landscape handle under the label text_string. The user can later use the jump command with text_string to set the current model handle and the current landscape handle back to the one stored under text_string. The user is prompted for verification if the same text_string is used in two setjump commands.

Separate setjump values are maintained for each session that is connected to the CLI Local Server. The setjump command retains information, that is, the session-assigned setjump text strings, only for the session that named it.

The command has the following format:

```
setjump [-n] <text_string>
```

-n

If the -n (no prompt) option is specified with the setjump command, then the system does not prompt if text_string has been used before.

- If setjump is entered with a new <text_string> and a current model exists, the following message is displayed:

  ```
  model <current_model_handle> and landscape
  <current_landscape_handle> stored under <text_string>
  ```

  where <current_model_handle> is the handle of the current model and <current_landscape_handle> is the handle of the current landscape.

- If setjump is entered with a new <text_string> and a current model does not exist, the following message is displayed:

  ```
  model undefined and landscape <current_landscape_handle>
  stored under <text_string>
  ```

- If setjump is entered with a text_string that has already been defined in a previous setjump command, the following message is displayed:

  ```
  setjump model: <text_string> already used. Overwrite?
  ```

  Valid responses are y, yes, Y, Yes, n, no, N, and No.

Example: setjump

```
$ current mh=0x400142
current model is 0x400142
current landscape is 0x400000
$ setjump -n tutorial
model 0x400142 and landscape 0x400000 stored under tutorial
```

More information:

current–Sets Model or Landscape (see page 36)
jump–Jumps to Saved Model or Landscape (see page 40)

show–Displays Object

To display objects, use the show command.

The command has the following format:

```
show models [mhr=low_model_handle-high_model_handle]
        [mth=model_type_handle][mname=model_name][lh=landscape_handle] |
devices [lh=landscape_handle]|
landscapes |
types [mthr=low_mth-high_mth] [mtname=mt_name]
                [flags=V|I|D|N|U|R] [lh=landscape_handle] |
relations [lh=landscape_handle] |
associations [mh=model_handle] |
parents [rel=relation] [mh=model_handle] |
children [rel=relation] [mh=model_handle] | attributes [-e]
                [attr=attribute_id[,iid=instance_id][,next]...|
                [attrr=low_attr-high_attr] [attrname=attr_name]]
                [mh=model_handle] |
attributes [-c] [-e]
                [attr=attribute_id[,iid=instance_id][,next]...|
                [attrr=low_attr-high_attr] [attrname=attr_name]]
                [mh=model_handle]|
attributes mth=model_type_handle [attrr=low_attr-high_attr]
                [attrname=attr_name]           [flags=E|R|W|S|T|G|O|M|D|P|L|V]
                [lh=landscape_handle] |
alarms [-a] [-x] [-t] [-s] [-d]
               [mh=model_handle|lh=landscape_handle] |
events [-x] [ -a | -n no_events ]
                [mh=model_handle|lh=landscape_handle] |
inheritance mth=model_type_handle [lh=landscape_handle] |
rules rel=relation [lh=landscape_handle] |
enumerations [attr=attribute_id] [mth=model_type_handle]
                [lh=landscape_handle] |
watch [mh=model_handle]
```

-a

If the -a (all) option is specified, show alarms do not perform any masking and displays all CRITICAL, MAJOR, MINOR, MAINTENANCE, SUPPRESSED, and INITIAL alarms.

-x

If the -x (expand) option is specified (and the variable $SPECROOT is set), the output of the show alarms command displays the text for the probable causes at the end of the output. The output of the show events command displays event formats. The number of characters displayed by the show events -x command is controlled by the .vnmshrc resource file parameter.

-d

If the -d option is specified, the output of the show alarms command displays the "Title" for every alarm listed. This option also displays all that the option -x displays.

max_show_event_length

If you are using a SpectroSERVER-only workstation to run CLI, the -x option does not provide the normal alarm cause or event format information because the CsPCause and CsEvFormat files do not exist in the SG-Support directory. Possible errors messages are:

- No cause information available (associated with show alarms)
- No event format information available (associated with show events)

To remedy this problem, copy the SG-Support/CsPCause and SG-Support/CsEvFormat directories and files to the <*$SPECROOT*>/SG-Support directory on the SpectroSERVER workstation.

Default: 512

-e

If the -e (enumerations) option is specified, the output of the show attributes command displays database enumeration strings.

-c

If the -c (Read Most Current) option is specified, the attribute Read Mode is set to Read Most Current. This mode uses the attribute value from the latest user interface poll, which is updated every 5 seconds. If this flag is not set, the Read Most Available mode is used. This mode uses the latest value stored in the database by the last CA Spectrum poll. Polling frequency is a user-defined interval.

-n

If the -n (number of events) option is specified, the output of the show events command displays the specified number of events.

-t

If the -t (trouble ticket id) option is specified, the output of the show alarms command displays the trouble ticket id field.

-s

If the -s (impact severity) option is specified, the ouput of the show alarms command displays the impact severity field.

For the show alarms and show events commands to work with the -x option, which displays probable cause messages for alarms and expanded event messages, OneClick must be installed on the local server, and the SPECROOT environment variable must be set to the path of the spectrum support root directory. For example, if the SG-Support files are in /usr/spectrum/SG-Support, set SPECROOT to /usr/spectrum.

More information:

The Startup File (see page 13)
current–Sets Model or Landscape (see page 36)
seek–Locates a Model (see page 41)

show alarm

The show alarms command shows all alarms for the model that is specified by model_handle. Or it shows only the most severe alarm (if the alarm is CRITICAL, MAJOR, or MINOR) for each model in the landscape that is specified by landscape_handle. If landscape_handle is specified, show alarms masks any models that have INITIAL, SUPPRESSED, or MAINTENANCE alarms. As a result, only models with CRITICAL, MAJOR, or MINOR alarms are displayed. If neither model_handle nor landscape_handle is specified, show alarms also performs masking and shows only the most severe alarm (if the alarm is CRITICAL, MAJOR, or MINOR) for each model in the current landscape.

The Ack field indicates whether the alarm has been acknowledged. The possible values for this field are Yes and No. The Stale field indicates whether an alarm is stale. The possible values for this field are Yes and No. The Assignment and Status fields show the alarm troubleshooter information and the alarm status, respectively. The alarm creation time is displayed in hh:mm:ss format.

The show alarms command displays information in the following format:

```
Id  Date       Time     PCauseId  MHandle  MName  MTypeName  Severity  Ack  Stale  Assignment  Status
id  mm/dd/yyyy  hh:mm:ss  cause_id  handle   name   name      severity  ack  stale  assignment  status
```

If show alarms is used with the -x option, a table of cause codes and probable cause text messages is displayed after the last alarm. For example:

0x10402 DUPLICATE PHYSICAL ADDRESS0x10302 SpectroSERVER has lost contact with this device.

Note: The show command displays a maximum of 16 characters for the model name. However, with the environment variable CLIMNAMEWIDTH, you can specify a different number of characters (up to 1024) to be displayed for model names.

Example: show alarms

The show alarms command displays information in the following format:

```
$ show alarms lh=0x110000
ID   Date        Time      PCauseId   MHandle     MName     MTypeName  Severity   Ack  Stale
Assignment  Status
928  05/11/2000  02:33:22  0x10c04    0x110000c   infinity  VNM        CRITICAL   No   No
McDonald    Working on it
```

More information:

Command Descriptions (see page 29)
CLI Environment Variables (see page 11)

show association

The show associations command shows all instantiated relations (associations) that are defined for the model with model_handle. If model_handle is not specified, show associations shows all instantiated relations for the current model.

The show associations command displays information in the following format:

```
LMHandle   LMName   Relation   RMHandle   RMName
handle     name     relation   handle     name
```

Example: show associations

The show associations command displays information in the following format:

```
$ show associations mh=0x400141

LMHandle   LMName     Relation        RMHandle   RMName
0x400001   LostFound  Lost_and_Found  0x400141   12.77-bridge
```

show attributes

The show attributes command shows the attributes specified by attr=attribute_id for the model with model_handle. If no attribute_id is specified, show attributes lists all attributes and their values for the model with model_handle.

If model_handle is not specified, show attributes shows all applicable attributes for the current model. You can specify a range of attributes using attr=low_attr-high_attr. The instance ID for an attribute can be specified in instance_id when displaying a single attribute or a list of attributes for a particular model. The instance_id must be a sequence of positive integers separated by periods. Instance IDs can only be specified for list attributes. List attributes are attributes that have the list flag set.

The following rules apply to list attributes:

- To display all attribute values and instance IDs for a list attribute, do not enter an instance_id with the attribute_id. Enter an attribute_id only.
- To display the first attribute value and instance ID for a list attribute, enter the following command after the attribute_id:

  ```
  ,next
  ```

- To display a specific attribute value and instance ID for a list attribute, enter an instance_id with the attribute_id.
- To display the next attribute value and instance ID after a specific instance ID of a list attribute, enter the following command after the instance_id:

  ```
  ,next
  ```

- An instance ID cannot be specified when displaying all the attributes of a model, for the following reasons:
  - An instance ID only applies to list attributes (for example, board and port attributes of a hub)
  - The instance ID for certain attributes of a model may differ from the instance ID of other attributes within the same model.
- The show attributes command shows all attributes (by ID, name, type, and flags) for model_type_handle in the landscape specified by landscape_handle. If landscape_handle is not specified, this command shows all model types that are defined in the current landscape. The Flags field lists the abbreviations of each attribute flags (separated by commas) that is currently set. If a flag is not set, its abbreviation is not inlcuded in the list.

  The following list includes the attribute flags and their abbreviations:
  - External = E
  - Readable = R
  - Writable = W
  - Shared = S
  - List = T
  - Guaranteed = G
  - Global = O

- Memory = M
- Database = D
- Polled = P
- Logged = L
- Preserve Value = V

Note: For a more detailed description of the attribute flags, see the *Model Type Editor User Guide*.

The show attributes command displays information in the following format:

```
Id     Name     Iid     Value
id     name     iid     value
```

The show attributes mth command displays information in the following format:

```
Id     Name     Type     Flags
id     name     type     flags
```

Example: show attributes

```
$ show attributes mh=0xcd00011

Id         Name                       Iid     Value
0xd0000    Modeltype_Name                     User
0x10000    Modeltype_Handle                   0x10004
0x10004    Contact_Status                     1
0x10009    Security_String                    ADMIN
0x1000a    Condition                          0

$ show attributes -e mh=0xcd00011

Id         Name                       Iid     Value
0xd0000    Modeltype_Name                     User
0x10000    Modeltype_Handle                   0x10004
0x10004    Contact_Status                     Established
0x10009    Security_String                    ADMIN
0x1000a    Condition                          Normal

$ show attributes -e attr=0x1000-0x11fff attrname=status mh=0xcd00011

Id         Name                       Iid     Value
0x10004    Contact_Status                     Established
0x110ed    Dev_Contact_Status                 2
0x111a56   ContactStatusEventSwitc            FALSE
```

```
$ show attributes attr=0x1006e mh=0x400165

Id         Name                        Iid     Value
0x1006e    Model_Name                          142.77

$ show attributes attr=0x100d4 mh=0x400165

Id         Name                        Iid     Value
0x100d4    If_Out_Ucast_Pkts            1      1169585
0x100d4    If_Out_Ucast_Pkts            2      1227557
0x100d4    If_Out_Ucast_Pkts            3      1227557
0x100d4    If_Out_Ucast_Pkts            4      8624873

$ show attributes attr=0x100d4,next mh=0x400165

Id         Name                        Iid     Value
0x100d4    If_out_Ucast_Pkts            1      1169589

$ show attributes attr=0x100d4,iid=2 mh=0x400165

Id         Name                        Iid     Value
0x100d4    If_Out_Ucast_Pkts            2      1227569

$ show attributes attr=0x100d4,iid=2,next mh=0x400165

Id         Name                        Iid     Value
0x100d4    If_out_Ucast_Pkts            3      1227573

$ show attributes mth=0x10004 lh=0xd00000

Id         Name                        Type            Flags
0xd0000    namingTree                  Group ID        S,D
0x10000    Modeltype_Name              Text String     R,S,M,K
0xd0200    upsBatteryCapacityInteger                   E,R

$ show attributes mth=0x3d0002 attrname=port

Id         Name                        Type            Flags
0x10023    Agent_Port                  Integer         R,W,M,D
0x112e3    IF_Port_Types               Octet String    R,W,S,D
0x11554    Create_IF_Port              Boolean         R,S,D
0x11d28    PortLinkDownEventCode       Counter         R,S,D
0x11d29    PortLinkUpEventCode         Counter         R,S,D
0x11d3d    support_ICMP                Boolean         R,W,D
0x11d41    Poll_Linked_Ports           Boolean         R,W,M,D
0x11e24    TelnetPortNum               Integer         R,W,G,D
```

```
$ show attributes mth=0x3d0002 attrname=port flags=rwmd

Id         Name                      Type            Flags
0x10023    Agent_Port                Integer         R,W,M,D
0x11d41    Poll_Linked_Ports         Boolean         R,W,M,D

$ show attributes -e attrname=port mh=0xcd00023

Id         Name                      Iid      Value
0x10023    Agent_Port                         161
0x112e3    IF_Port_Types                      11.0.22.0
0x11554    Create_IF_Port                     TRUE
0x11d28    PortLinkDownEventCode              66312
0x11d29    PortLinkUpEventCode                66313
0x11d3d    support_ICMP                       TRUE
0x11d41    Poll_Linked_Ports                  TRUE
0x11e24    TelnetPortNum                      0
```

show children

The show children command shows the children in relation to the model with model_handle. If relation is not specified, show children shows the children in all relations. If model_handle is not specified, the command shows the children for the current model.

The show children command displays information in the following format:

```
MHandle    MName    MTypeHn   MTypeName   Relation
handle      name     handle    name        relation
```

Example: show children

```
$ show children mh=0x400009

MHandle    Name    MTypeHnd   MTypeName      Relation
0x40000d   12.84   0x100d6    Bdg_CSI_CNB2   Collects
```

show devices

The show devices command shows a listing of all device models in the landscape that is specified by the landscape_handle.

The show devices command displays information in the following format:

```
MHandle   MName   MTypeHnd   MTypeName
Handle    Name    Handle     Name
```

Example: show devices

```
$ show devices lh=0x400000

MHandle     MName           MTypeHnd    MTypeName
0x1005c0    10.253.32.101   0x3d002     GnSNMPDev
0x100030    10.253.2.10     0x2c60021   RstonesSwRtr
```

show enumerations

The show enumerations command shows enumerated string value mapping for the corresponding enumerated value specified.

The show enumerations command displays information in the following format:

```
Id   String   Value
id   string   value
```

The show enumerations mth command displays information in the following format:

```
MHandle   String   Value
HandLE    string   value
```

Example: show enumerations

```
$ show enumerations attr=0x10004

ID        String        Value
0x10004   Lost          0
0x10004   Established   1
0x10004   INITIAL       2

$ show enumerations mth=0x10004

ID        String        Value
0x10004   Lost          0
0x10004   Established   1
0x10004   INITIAL       2
```

show events

The show events command shows the events for the model with model_handle or the events for all models in the landscape that is specified by landscape_handle. By default the show events command shows the 2,000 most recent events for the model that is specified by model_handle or landscape_handle. If the -a option is specified, this command shows a maximum of 10,000 events for the model which is specified by model_handle or landscape_handle.

If the -n option is specified with an explicit no_events statement, the specified number of events is displayed for the model which is specified by model_handle or landscape_handle. If neither model_handle nor landscape_handle is specified, this command shows events for all models in the current landscape. If the -x option is specified, the CLI displays text messages explaining the event types. The event time is displayed in hh:mm:ss format.

The show events command displays information in the following format:

```
Date          Time        Type     MHandlE    MName    MTypeName
mm/dd/yyyy    hh:mm:ss    type     handle     name     name
```

If show events is used with the -x option, the events displayed do not have a fixed format. The following is an example of typical output:

```
Thur 11 May, 2000 - 8:04:01 - Alarm number 10 generated for device AntLAN of type
LAN_802_3.
Current condition is INITIAL(DEFAULT).
(event [00010701])
```

Example: show events

```
$ show events lh=0x400000

Date         Time       Type      MHandle    MName    MTypeName
04/25/1999   13:27:38   0x10302   0x4000f9   1.3      Host_IBM
04/25/1999   13:27:38   0x10202   0x400131   qa1sgi   Host_SGI

$ show events -n

Date         Time       Type      MHandle             MName             MTypeName
08/21/1999   11:30:02              0x100090xcd00067    els100-01.india   RMONApp
08/21/1999   11:25:33              0x100090xcd00067    els100-01.india   RMONApp
08/21/1999   11:20:17              0x100090xcd00067    els100-01.india   RMONApp
08/21/1999   11:15:52              0x100090xcd00067    els100-01.india   RMONApp
08/21/1999   11:10:27              0x100090xcd00067    els100-01.india   RMONApp
```

show inheritance

The show inheritance command shows the model type inheritance for the model type that is specified by model_type_handle in the landscape that is specified by landscape_handle. If the landscape_handle is not specified, the current landscape is used. The possible values for this field are Base or Derived.

The show inheritance command displays information in the following format:

```
MHandle   MName   Flags   Inheritance
handle    name    flags   inheritance
```

Example: show inheritance

```
$ show inheritance mth=0x1037b lh=0x400000

Handle    Name        Flags    Inheritance
0x10000   Root        V,D      Base
0x103ad   BanVinesFS  V,I,U    Derived
```

show landscapes

The show landscapes command shows all landscapes that are defined for each SpectroSERVER. The landscape map that is displayed is the map of the initial SpectroSERVER.

The show landscapes displays information in the following format:

```
SSName    Precedence    Port    Service    LHandle
ssname    precedence    port    service    handle
```

Example: show landscapes

```
$ show landscapes

SSName    Precedence    Port    Service   LHandle
devsgi    10            0xbeef  0x10101   0x28000000
devibm    10            0xbeef  0x10101   0x11f00000
```

show models

The show models command shows all models that are defined in the landscape, which is specified by landscape_handle. If landscape_handle is not specified, this command shows all models that are defined in the current landscape. A range of model handles can be specified by the following command:

```
mhr=low_model_handle-high_model_handle
```

Specific models can be searched for by specifying mname=model_name.

User models are identified by the show models command as either (Active) or (Not Active). If the user model status is (Not Active), the user cannot yet connect to the server. Once the user model status is (Active), the user can connect to the server.

The show models command displays information in the following format:

```
MHandle     MName     MTypeHnd     MTypeName
handle      name      handle       name
```

Example: show models

```
$ show models lh=0x400000

MHandle    MName    MTypeHnd    MTypeName
0x400004   World    0x10040     World
0x4000d9            0x10020     AUI

$ show models mname=

MHandle     MName        MTypeHnd        MTypeName
0xcd00016   0x1120002    AppDataServer
0xcd00022   0x1006b      SnmpPif
0xcd00030   0x1028f      IcmpPif

$ show models mhr=0xcd00000-0xcd000ff mth=0x230018 mname=india lh=0xcd00000

MHandle     MName                        MTypeHnd    MTypeName
0xcd000a3   hplaser.zeitnet.India.com    0x230018    TCP2_App
0xcd0002b   desire.zeitnet.India.com     0x230018    TCP2_App
```

show parents

The show parents command shows the parents in relation to the model with model_handle. If relation is not specified, it shows the parents in all relations. If model_handle is not specified, show parents shows the parents for the current model.

The show parents command displays information in the following format:

```
MHandle    MName    MTypeHnd    MTypeName    Relation
handle     name     handle      name         relation
```

Example: show parents

```
$ show parents mh=0x40000d

MHandle    MName              MTypeHnd    MTypeName      Relation
0x400009   auto-lan-30x1003c              LAN_802_3    Collects
```

show relations

The show relations command shows all relations that are currently defined in the landscape specified by landscape_handle. If landscape_handle is not specified, this command shows all relations that are defined in the current landscape.

The show relations command displays information in the following format:

```
Name             Type
relation_name    relation_type
```

Example: show relations

```
$ show relations

Name Type
Passes_Through MANY_TO_MANY
Lost_and_Found ONE_TO_MANY
Owns ONE_TO_MANY
Contains ONE_TO_MANY
```

show rules

The show rules command shows the rules for a relation. The relation is specified in the landscape that is specified by landscape_handle. If landscape_handle is not specified, the current landscape is used.

The show rules command displays information in the following format:

```
LMTHandle    LMTName    RMTHandle    RMTName
handle       name       handle       name
```

Example: show rules

```
$ show rules rel=Owns lh=0x400000

LMTHandle    LMTName    RMTHandle    RMTName
0x102da      Org_Owns   0x10043      Site
0x102da      Org_Owns   0x210023     Rtr_CiscoMGSshow
```

show types

The show types command shows all model types that are currently defined in the landscape that is specified by landscape_handle. If landscape_handle is not specified, this command shows all model types defined in the current landscape. The Flags field lists the abbreviations for each of six attribute flags that are currently set. If a flag is not set, its abbreviation is not included in the list.

The following list includes the model type flags and their abbreviations:

- Visible = V
- Instantiable = I
- Derivable = D
- No Destroy = N
- Unique = U
- Required = R

The show types command [mth=low_mth-high_mth] shows all model types within the range between low_mth and high_mth.

Note: For more information about model type flags, see *Model Type Editor User Guide.*

The show types command displays information in the following format:

```
Handle    Name    Flags
handle    name    flags
```

Example: show types

```
$ show types lh=0x400000

Handle    Name          Flags
0x10000   Root          V,D
0x10080   Gen_Rptr_Prt  V,D

$ show types mthr=0x10002-0x10008

Handle    Name             Flags
0x10002   Network_Entity
0x10003   VNM              V,I,D,N,U,R
0x10004   User             V,I,D
0x10005   VIB
0x10007   DataRelay        V,D

$ show types mthr=0x210020-0x21002f mtname=Rtr_Cisco lh=0xcd00000

Handle      Name             Flags
0x210020    Rtr_CiscoAGS     V,I,D
0x210021    Rtr_CiscoCGS     V,I,D
0x210022    Rtr_CiscoIGS     V,I,D
0x210023    Rtr_CiscoMGS     V,I,D
0x210024    Rtr_CiscoMIM     V,I,D
0x21002b    Rtr_Cisco2500    V,I,D
0x21002c    Rtr_CiscoMIM3T   V,I,D
0x21002d    Rtr_Cisco3000    V,I,D
0x21002e    Rtr_Cisco4000    V,I,D
0x21002f    Rtr_Cisco7000    V,I,D
```

```
$ show types flags=VIDNUR lh=0xcd00000

Handle        Name              Flags
0x25e0000     MgmtInventory     V,I,D,N,U,R
0x10040       World             V,I,D,N,U,R
0x102cf       Top_Org           V,I,D,N,U,R
0x10003       VNM               V,I,D,N,U,R
0x102be       LostFound         V,I,D,N,U,R
0x25e0001     TopologyWrkSpc    V,I,D,N,U,R
0x10091       Universe          V,I,D,N,U,R
```

show watch

The show watch command lists applicable watch data for a model that is specified by model_handle.

The show commands use the following defaults when landscape_handle and model_handle are not specified:

Command	Default
show alarms	current landscape
show associations	current model
show attributes	current model
show attributes mth	current landscape
show children	current model
show devices	current landscape
show enumerations	current landscape
show enumerations mth	current landscape
show events	current landscape
show inheritance	current landscape
show models	current landscape
show parents	current model
show relations	current landscape
show rules	current landscape
show types	current landscape
show watch	current model

Note: The ‘show alarms’ and ‘show events’ commands can work with the x option to display probable cause messages for alarms and expanded event messages.

Verify the following prerequisites:

- OneClick is installed on the local server.
- The environment variable SPECROOT is set to the path of the root directory (SG-Support).

  For example, if the SG-Support files are in/usr/spectrum/SG-Support, set SPECROOT to /usr/spectrum.

The show watch command displays information in the following format:

```
Watch_Id   Watch_Name   Watch_Type   Watch_Status
watch_id   watch_name   watch_type   watch_status
```

Example: show watch

```
$ show watch mh=0xc600015

Watch_Id     Watch_Name   Watch_Type   Watch_Status
0xffff0001   watch798     Calc         Active
```

stopShd–Terminates CLI Local Server

Use the stopShd command to terminate the CLI Local Server (VnmShd daemon). The stopShd command disconnects all CA Spectrum CLI users from the currently connected SpectroSERVER and terminates the CLI Local Server. This command prompts the user for confirmation before disconnecting users and shutting down the server. (You can also shut down the daemon by using the kill -2 command.)

The command has the following format:

```
stopShd [-n]
```

-n

Specifies 'no prompt'. Include this option with the stopShd command to disable confirmation prompts.

Otherwise, the following message is always displayed:

```
stopShd: n users are connected, are you sure?
```

The 'n' represents the number of connected users, including yourself.

Valid responses are y, yes, Y, Yes, n, no, N, No.

If the command is successful, the following message is displayed:

```
stopShd: successful
```

When stopShd terminates the CLI Local Server, the following message is displayed on the system console:

```
VnmShd: stopShd executed. Exiting...
```

Example: stopShd

```
$ stopShd
stopShd: 2 users are connected, are you sure? y
stopShd: successful
```

More information:

disconnect–Disconnects from SpectroSERVER (see page 39)

update–Updates Model and Model Attributes

Use the update command to update model and model type attributes.

The command has the following format:

```
update  [mh=modelhandle]
attr=attribute_id[,iid=instance_id],val=value
                [attr=attribute_id[,iid=instance_id],val=value...] |
                [mh=modelhandle]
                attr=attribute_id,iid=instance_id,remove
                [attr=attribute_id,iid=instance_id,remove...] |
                [-n] mth=model_type_handle |
attr=attribute_id,val=value [attr=attribute_id,val=value ... ]
                [lh=landscape_handle] |
alarm [-r] aid=alarm_id <assign=troubleshooter |
                status=status_text | ticket=troubleticketID |
                ack=(true|false)> [lh=landscape_handle] |
action=action_code [watch=watch_id] [mh=modelhandle]
```

-n

If the -n (no prompt) option is specified with the update command, then the system does not prompt for confirmation. This option is useful in CLI scripts.

-r

The -r (replace status text/replace trouble ticket ID) option can be specified with the update alarm command when using the status or the ticket arguments. When the -r option is used, the existing alarm status text or alarm trouble ticket ID is replaced with the text specified by the status argument or the ticket argument. When the -r option is not used, the new values are appended to the existing values.

action_code

- reconfig, 0x1000e, or 65550 to reconfigure a model
- activate, 0x00480003, or 4718595 to activate a watch
- deactivate, 0x00480004, or 4718596 to deactivate a watch
- reconfigure_apps, 0x210008, or 2162696 to reconfigure application models on Cisco and Wellfleet devices
- reload_event_disp, 0x000100a2, or 65698 to update the SpectroSERVER with changes to EventDisp and AlertMap files

Note: The watch = *<watch_id>* parameter is applicable only for the following actions: activate (or the hexadecimal equivalent 0x00480003) and deactivate (or the hexadecimal equivalent 0x00480004).

The following points describe the features of update command:

- The update command updates the attribute specified by attribute_id value either for model with model_handle or for all models with the model type model_type_handle in the landscape specified by landscape_handle.
- Multiple attributes can be updated with one update command by specifying multiple attribute_id, value pairs, each pair that is separated from adjacent pairs by a space.
- The remove option removes instances that are specified from a list attribute.
- If landscape_handle is not specified when updating model type attributes, the current landscape is used. If model_handle is not specified, then the specified attribute of the current model is updated.
- When you are updating model type attributes, remember that only shared attributes can be updated. Shared attributes are attributes that have the shared flag set. Use the show attributes command to see if an attribute is shared.
- Security-sensitive attributes, such as User_Community_String and Model_Security_String, can be updated through CLI. However, the current user model cannot update its own User_Security_String or Security_String, but it can update those of other models.
- The update command also lets the user specify an instance ID when changing a single attribute value. When updating a list of attribute values, an instance ID can be specified for each attribute on the list. instance_id is the instance ID for the corresponding attribute. The instance_id must be a positive integer, or sequence of dot-separated positive integers.
- If an instance ID is not specified, the update command uses the first valid instance that it finds for the attribute. If no valid instances are found, an error message is displayed:

  ```
  update: no valid instance for list attribute <attr_id>
  ```

- The update alarm command updates the alarm specified by alarm_id with the value specified by the Troubleshooter (Troubleshooter model handle or Troubleshooter name), status_text, troubleticketID, or ack parameter. To clear the existing alarm values for Troubleshooter, Status text, or Trouble Ticket ID, you can set the appropriate parameter to have no value (status=, ticket=, or assign=). The landscape_handle parameter specifies the landscape in which the alarm will be found.

- The update action command performs an action specified by action_code on a device specified by model_handle. With action_code reconfig, any device of model type GnSNMPDev, or of any model type that inherits from GnSNMPDev, can be reconfigured. The activate or deactivate action_code update a watch status on a device of a specified model_handle. When the activate action object is sent, there may be a short delay between the time the watch status changes from INITIAL to ACTIVE, depending upon the intelligence that is built into the selected model. The watch_id of the watch slated to have its status updated can be obtained by using the show watch command. The reconfigure_apps action_code update application model types for Cisco and Wellfleet device models. The reload_event_disp action_code update the SpectroSERVER with changes made to EventDisp or AlertMap files.
- Use caution when using the update action command. As with any CLI command, you can corrupt the SpectroSERVER database if you use this command incorrectly. For example, inadvertently reconfiguring a critical router can cause unpredictable results on your network.
- If update is entered with a valid model_handle or valid model_type_handle, valid attribute_id(s), and valid values, the modified attributes and their values are displayed in the following format:

  ```
  Id  Name    Value
  Id  Name    Value
  ```

- If you do not use the -n option when updating models of a specified model type, the following confirmation message is displayed:

  ```
  update: all models of this type will be updated, are you sure?
  Valid responses are y, yes, Y, Yes, n, no, N, No.
  ```

- If the update alarm command is successful, the following message is displayed:

  ```
  update:successful
  ```

- If the update action command is successful, the following message is displayed:

  ```
  update action: successful
  ```

Examples: Update

- In the following example, the update command with an instance_id is used to disable port 7 on board 5 of the Hub represented by model handle 0x4001f6:

  ```
  $ update mh=0x4001f6 attr=0x10ee0,iid=5.7,val=1
  Id        Name              Iid       Value
  0x10ee0   CsPortAdminState             1
  ```

- In the following example, the update command is used with the remove option to remove an IP address (iid) from the deviceIPAddressList (attr) for a particular model (mh).

  ```
  $ update mh=0xc600018 attr=0x12a53,iid=10.253.8.65,remove
  update: successful
  ```

- In the following example, the update command is used to update the attribute named AutoPlaceStartX on all models of the model type represented by model type handle 0x10059.

```
$ update mth=0x10059 attr=0x118f2,val=100 lh=0x400000
update: all models of this type will be updated, are you sure? y
Id    Name              Value
Id    AutoPlaceStartX   100
```

- In the following example, the update alarm command is used to update an alarm troubleshooter assignment.

```
$ update alarm aid=928 assign=0xa600722
update: successful
```

- In the following example, the update alarm command is used to update alarm status. The -r option is used to overwrite the existing status.

```
$ update alarm -r aid=928 status='Working on it'
update: successful
```

- In the following example, the update alarm command is used to update the alarm Trouble Ticket ID. The -r option is used to overwrite the existing value for Trouble Ticket ID.

```
$ update alarm -r aid=928 ticket='Ax1009'
update: successful
```

- In the following example, the update alarm command is used to clear the existing value for Trouble Ticket ID.

```
$ update alarm aid=928 ticket=
update: successful
```

- In the following example, the update alarm command is used to acknowledge the alarm.

```
$ update alarm aid=928 ack=TRUE
update: successful
```

- In the following example, the update command is used to restrict updating of the User_Community_String.

```
$ update mh=0x9a000ff attr=0x1007a,val=AA,11
update: successful
```

- In the following example, the update action command is used to reconfigure a Cisco router.

```
$ update action=reconfig mh=0xc600030
update action: successful
```

```
$ update action=activate watch=0xffff0001 mh=0xc600015
```

```
Watch_Id      MHandle      Watch_Status
0xffff0001    0xc600015    INITIAL
```

```
$ update action=0x480004 watch=0xffff0001 mh=0xc600015
```

```
Watch_Id      MHandle      Watch_Status
0xffff0001    0xc600015    INACTIVE
```

Appendix A: Sample Scripts

Sample Scripts Overview

The sample scripts included with CLI demonstrate how CLI commands can be incorporated into UNIX shell scripts so that you can automate your CLI sessions. You can find some of these scripts, or some of the functions within them, useful for your own work.

CLI includes the following scripts, and a readme file that describes the scripts in the <*$SPECROOT*>/vnmsh/sample_scripts directory:

- active_ports
- app_if_security
- cli_script
- database_tally
- update_mtype
- octet_to_ascii.pl

Review the following prerequisites when you work with CLI scripts:

- Each script has an internal variable named CLIPATH. To use a script, set the CLIPATH variable to the pathname of the directory where CLI executables are located.
- The CLIPATH variable and the other environment variables that are pathnames can be *full* or *relative* pathnames depending on how the script is run. Use *full* pathnames for the CLIPATH and other environment variables when you run a sample script as a cron script. Otherwise, you can use *relative* pathnames for these variables.
- Except for update_mtype, you can run all the CLI scripts as cron scripts.

  Note: Do not run update_mtype as a cron script because it prompts the user for input.
- When you run CLI scripts, specify the correct names for the vnm_hostname variable in the .vnmshrc file.

active_ports Script

Use the active_ports script to identify all ports for each board of an IRM2 hub and to identify the active ports on each board.

The active_ports script places a report for the hub with the name hub_name in a file with the name output_file. This report lists all the ports for each board. An asterisk (*) in the ON column of the report shows you which ports are active.

This script has the following format:

```
active_ports <hub_name> <output_file>]
```

app_if_security Script

Use the app_if_security script to update the Security_String attribute value in all the interface and application models in the CA Spectrum database. The app_if_security script does update by copying the attribute value from the parent model. The script does not update any models if the recipient model (child) already has a value for the Security_String attribute or if the parent does *not* have a Security_String attribute value. After updating a models security string, administrators can use this script to update the security string of the models children.

This script has the following format:

```
app_if_security
```

cli_script Script

Use cli_script to execute most of the CLI commands in batch mode when you provide a data file as input. The CLI sample data file, named datafile, contains switches that indicate the command to execute and also the necessary parameters to pass to the command. The script verifies that each command is executed successfully and also maintains a runtime log.

One advantage of this script is that you can create batch files using names instead of handles. For example, you can use a model type name, rather than the hexadecimal model type handle. While this makes the files easier to create and read, the real advantage comes when you want to perform subsequent actions on a model that you have created. Instead of assigning hexadecimal model handles to the model, you can refer to the model by name.

This script has the following format:

```
cli_script datafile
```

The cli_script uses two files, datafile and clean.awk, that are also located in the sample_scripts directory.

datafile

Contains the input for cli_script. It contains each CLI command currently implemented in cli_script. See the cli_script header information for instructions about the format and syntax of this file.

clean.awk

Contains the input used in execution. The .awk files are used for formatting what data appears to the console.

Consider the following points when working with cli_script:

- Remember to change the “dummy” Network_Address (255.255.255.255) in the sample datafile to a real address.
- If you move the cli_script to another directory, you must update the environment variable SPECROOT to the support root directory (SG-SUPPORT).

  For example, if the SG-SUPPORT files are in /usr/spectrum/SG-Support, set SPECROOT to /usr/spectrum.

database_tally Script

Use this script to determine how many models of each type are currently in the database. Administrators may find this script useful when evaluating system performance. The script displays a list of all the model types and the number of models of each model type in the database.

This script has the following format:

`database_tally <vnm-name>`

update_mtype Script

Use this script to update a specific attribute for all models of a model type. If the attribute is a shared attribute of the model type, the script does not update the model's attribute. One advantage of this script is that you can use the model and attribute names at the prompt rather than their hexadecimal ID handles.

Note: Set the CLIMNAMEWIDTH environment variable to 256. The high value prevents truncation of model names that can cause false matches when running the script.

This script has the following format:

update_mtype *<model_name>* \ *<model_typename>*[*<attribute_name>* | *<attribute_id>* *<value>*]

model_name | model_type_name

Specifies a model name, or part of a model name, for a model of the model type for which the attribute update is done. You can specify any model of the model type in the command.

The script then displays a listing of all model types that have models with names containing the model name argument that you entered. The script asks you to select a model type from the list.

If you use the model name instead of the model type name, the script updates all models whose names include the string entered on the command line or at the prompt. In this case, all models of a given model type are not updated as described above.

Note: We recommend using the model name or model type name at the prompt and not handle.

attribute_name | attribute_id

If you do not specify these arguments initially, the script prompts for attribute name or attribute id, when it runs. At this point, you must specify either the attribute name or part of the attribute name. The script then asks you to select from a list of attributes containing the text that you entered. You can run the entire script, therefore, without prior knowledge of the hexadecimal model type handles or attribute handles.

More information:

CLI Environment Variables (see page 11)

active_ports Script

Use this script to convert Octet_String format XX.YY.ZZ to its ASCII string representation.

Appendix B: Error Messages

ack alarm: <alarm_id>: invalid alarm id

Reason:

The alarm ID that you entered is invalid.

Action:

Enter the ack alarm command again, using a valid alarm_id.

ack alarm: <landscape_handle>: invalid landscape handle

Reason:

The landscape handle that you entered for the alarm is invalid.

Action:

Enter the ack alarm command again, using a valid landscape_handle.

command: failed to connect with VnmShd, please run connect first

Reason:

You attempted to run other commands before running the connect command.

Action:

Begin a CLI session with the connect command.

connect: already connected to <hostname> since <date/time>

Reason:

The attempt to connect is unnecessary. You are already connected to a SpectroSERVER host.

Action:

None.

connect: cannot open resource file <pathname>/.vnmshrc

Reason:

The connect command cannot find the CLI resource file .vnmshrc.

Action:

The .vnmshrc resource file must be in the same directory as the connect command itself.

More information:

The Startup File (see page 13)

connect: can only connect to SpectroSERVERs in <hostname> landscape map - other user(s) already connected

Reason:

The connect command has already been used to connect to a particular SpectroSERVER. You can connect only to a SpectroSERVER that is in the landscape map of the original SpectroSERVER.

Action:

None

connect: ERROR: No such CA Spectrum user as <username>

Reason:

The first user of the connect command is not defined as a CA Spectrum user.

Action:

Reconnect to the SpectroSERVER as a CA Spectrum user.

connect: <hostname> not responding or not permitting access

Reason:

The connect command cannot connect to SpectroSERVER because the hostname is incorrect, the SpectroSERVER is not running, or the user has no user model.

Action:

Verify that the hostname is correct, that SpectroSERVER is running, and that the user has a user model.

connect: <landscape_handle>: invalid landscape handle

Reason:

The landscape_handle specified by the user is not valid for the specified hostname, or the handle cannot be accessed by your VNM.

Action:

Verify that the landscape_handle is valid for the specified hostname, and verify that the handle can be accessed by your VNM.

connect: incompatible SpectroSERVER <version>

Reason:

The user is attempting to connect to a SpectroSERVER host whose version is incompatible with the CLI version.

Action:

Update the version of CLI that you are using.

connect: invalid <value> for CLISESSID

Reason:

The connect command is used within a cron script or the windowing system returns 0 for ttyslot and the environment variable CLISESSID is set to a non-numeric value.

Action:

Use the connect command outside of a cron script and set CLISESSID to a numeric value.

connect: variable <CLISESSID> not set

Reason:

You have attempted to use the connect command within a cron script without first setting the CLISESSID environment variable.

Action:

When using connect within a cron script, set the environment variable CLISESSID.

create: user not permitted to create alarm

Reason:

You are not permitted to create an alarm.

Action:

Verify your user permissions.

create: user not permitted to create association

Reason:

You are not permitted to create an association.

Action:

Verify your user permissions.

create: user not permitted to create event

Reason:

You are not permitted to create an event.

Action:

Verify your user permissions.

create: user not permitted to create model

Reason:

You are not permitted to create a model.

Action:

Verify your user permissions.

create alarm: <probable_cause_id>: invalid alarm cause

Reason:

The create alarm command was entered with an invalid probable_cause_id.

Action:

Re-enter the create alarm command with a valid probable_cause_id.

create alarm: <alarm_severity>: invalid alarm severity

Reason:

The create alarm command was entered with an invalid alarm_severity.

Action:

Re-enter the create alarm command with a valid alarm_severity.

create alarm: <model_handle>: invalid model handle

Reason:

The create alarm command was entered with an invalid model_handle.

Action:

Re-enter the create alarm command with a valid model_handle.

create association: <left_model_handle>: invalid model handle

Reason:

The create association command was entered with an invalid left_model_handle.

Action:

Re-enter the create association command with a valid left_model_handle.

create association: models belong to different landscapes

Reason:

The create association command was entered with a left_model_handle and a right_model_handle in different landscapes.

Action:

Use the same landscape for both handles.

create association: rel=<relation>: invalid relation

Reason:

The create association command was entered with an invalid relation.

Action:

Re-enter the create association command with a valid relation.

create association: <right_model_handle>: invalid model handle

Reason:

The create association command was entered with an invalid right_model_handle.

Action:

Re-enter the create association command with a valid right_model_handle.

create event: <event_type>: invalid event type

Reason:

The create event command was entered with an invalid event_type.

Action:

Re-enter the create event command with a valid event_type.

create event: <landscape_handle>: unknown landscape

Reason:

The create event command was entered with an invalid landscape_handle.

Action:

Re-enter the create event command with a valid landscape_handle.

create event: <model_handle>: invalid model handle

Reason:

The create event command was entered with an invalid model_handle.

Action:

Re-enter the create event command with a valid model_handle.

create model: <attribute_id>: invalid attribute id

Reason:

No model is created because the create model command was entered with an invalid attribute_id.

Action:

Re-enter the create model command with a valid attribute_id.

create model: DCM device unreachable

Reason:

No model was created because the create model command was entered with an invalid ip_address. The DCM (Device Communication Manager) issues this error message.

Action:

Re-enter the create model command with a valid ip_address.

create model: Device limit exceeded

Reason:

No model was created because the Branch Manager SpectroSERVER (50 device model limit) or the Site Manager SpectroSERVER (250 device model limit) contains the maximum number of device models it can contain.

Action:

Verify that the number of device models on the SpectroSERVER have not met the prescribed limits. If they have, you may need to delete some and then re-enter the create model command.

create model: <landscape_handle>: invalid landscape handle

Reason:

No model was created because the create model command was entered with an invalid landscape_handle.

Action:

Re-enter the create model command with a valid landscape_handle.

create model: <model_type_handle>: invalid model type handle

Reason:

No model is created because the create model command was entered with an invalid model_type_handle.

Action:

Re-enter the create model command with a valid model_type_handle.

create model: <value>: invalid value

Reason:

No model is created because the create model command was entered with an invalid value.

Action:

Re-enter the create model command with a valid value.

create model: <value>: No community name

Reason:

The community name provided in the create request was incorrect.

Action:

Re-enter the create command with a valid community name.

current: <model_handle>: invalid model handle current model is <current_model_handle>

Reason:

An invalid model_handle was specified so the current model and the current landscape are unchanged.

Action:
Re-enter a valid model_handle if you want to modify the current model and current landscape.

current: <landscape_handle>: invalid landscape handle current landscape is <current_landscape_handle>

Reason:

Since an invalid landscape_handle was specified, the current model and the current landscape are unchanged.

Action:

Re-enter a valid landscape_handle if you want to modify the current model and current landscape.

current: <landscape_handle>: not responding or not permitting access current model is <current_model_handle>

Reason:

A landscape_handle was specified and the OneClick for the landscape was down or the user did not have a user model on that landscape.

Action:

Verify that the OneClick for the landscape in question is running; verify that you have a user model on the landscape in question; and then re-enter the landscape_handle.

current: <landscape_handle>: not responding or not permitting access current landscape is <current_landscape_handle>

Reason:

A model_handle was specified and the SpectroSERVER for the landscape containing that model was down or the user did not have a user model on that landscape.

Action:

Verify that the SpectroSERVER for the landscape in question is running; verify that you have a user model on the landscape in question; and then re-enter the model_handle.

destroy: user not permitted to destroy alarm

Reason:

You are not permitted to destroy an alarm.

Action:

Verify your user permissions.

destroy: user not permitted to destroy association

Reason:

You are not permitted to destroy an association.

Action:

Verify your user permissions.

destroy: user not permitted to destroy model

Reason:

You are not permitted to destroy a model.

Action:

Verify your user permissions.

destroy alarm: aid=<alarm_id>: invalid alarm id

Reason:

The destroy alarm command was entered with an invalid alarm_id.

Action:

Re-enter the destroy alarm command with a valid alarm_id.

destroy alarm: <landscape_handle>: invalid landscape handle

Reason:

The destroy alarm command was entered with an invalid landscape_handle.

Action:

Re-enter the destroy alarm command with a valid landscape_handle.

destroy association: rel=<relation>: invalid relation

Reason:

The destroy association command was entered with an invalid relation.

Action:

Re-enter the destroy association command with a valid relation.

destroy association: <left_model_handle>: invalid model handle

Reason:

The destroy association command was entered with an invalid left_model_handle.

Action:

Re-enter the destroy association command with a valid left_model_handle.

destroy association: <right_model_handle>: invalid model handle

Reason:

The destroy association command was entered with an invalid right_model_handle.

Action:

Re-enter the destroy association command with a valid right_model_handle.

destroy association: association does not exist between given models

Reason:

An attempt was made to destroy an association between two models that do not exist.

Action:

Verify the existence of the two models belonging to the association you are attempting to destroy.

destroy association: models belong to different landscapes

Reason:

The destroy association command was entered with a left_model_handle and a right_model_handle in different landscapes.

Action:

Re-enter the destroy association command using a left_model_handle and a right_model_handle from the same landscape.

destroy model: <model_handle>: invalid model handle

Reason:

The destroy model command was entered with an invalid model_handle.

Action:

Re-enter destroy model command with a valid model_handle.

disconnect: failed

Reason:

The disconnect command failed.

Action:

Re-try the disconnect command.

disconnect: failed to connect with VnmShd, please run connect first

Reason:

An attempt was made to run disconnect when the CLI Local Server was not running.

Action:

None. You are already disconnected.

disconnect: not connected

Reason:

The disconnect command failed since the user was not connected to the SpectroSERVER.

Action:

None. You are already disconnected.

jump: <text_string>: text string not defined

jump:<text_string>: text string not defined

`where text_string1, text_string2... are the currently defined text strings.`

Reason:

The jump command was entered with an undefined text_string.

Action:

Re-enter the jump command using any of the defined text strings.

<pathname>/VnmShd: not found

<pathname>/VnmShd: not found

`connect: failed`

`where pathname represents the path to the directory in which CLI attempted to execute VnmShd.`

Reason:

The connect command cannot find the CLI Local Server.

Action:

Make sure VnmShd and connect are in the same directory and then re-enter the connect command.

Please connect first

Reason:

After connect executed, you ran disconnect or stopShd and then attempted to run another command.

Action:

Reissue the connect command first.

seek: <attribute_id>: invalid attribute id

Reason:

The seek command was entered with an invalid attribute_id.

Action:

Re-enter the seek command with a valid attribute_id.

seek: <error>: attribute not keyed

Reason:

The seek command was entered with the attribute_id of an attribute that was not keyed.

Action:

Re-enter the seek command with an attribute_id of a keyed attribute.

seek: <value>: invalid value

Reason:

The seek command was entered with an invalid value.

Action:

Re-enter the seek command with a valid value.

show attributes: <attribute_id>: non list attribute

Reason:

The show attributes command was entered with an instance_id for a non list attribute_id.

Action:

Re-enter the show attributes command with an instance_id for a list attribute_id.

show attributes: <attribute_id>: invalid attribute id

Reason:

The show attributes command was entered with an invalid attribute_id.

Action:

Re-enter the show attributes command with a valid attribute_id.

show attributes: <instance_id>: invalid instance id

Reason:

The show attributes command was entered with an invalid instance_id. An instance_id is invalid if it does not consist of a sequence of non-negative integers or if it does not exist for the specified attribute.

Action:

Re-enter the show attributes command with a valid instance_id.

show attributes: <model_type_handle>: invalid model type handle

Reason:

The show attributes command was entered with an invalid model_type_handle.

Action:

Re-enter the show attributes command with a valid model_type_handle.

show: <landscape_handle>: invalid landscape handle

Reason:

A show command that uses an optional landscape_handle was entered with an invalid landscape_handle.

Action:

Re-enter the show command with a valid landscape_handle.

show: <model_handle>: invalid model handle

Reason:

A show command that uses an optional model_handle was entered with an invalid model_handle.

Action:

Re-enter the show command with a valid model_handle.

show: no current model defined

Reason:

A show associations command that uses an optional model_handle was entered but no model_handle was specified and no current model was defined.

Action:

Re-enter the show associations command, including both a model_handle and current model.

show alarms: no cause information available

Reason:

The show alarms command was used with the -x option, and the CA Spectrum alarm files containing the probable cause text messages are not available.

Action:

For the show alarms command to work with the -x option, which displays probable cause messages for alarms and expanded event messages, OneClick must be installed on the local server, and the environment variable SPECROOT must be set to the path of the Support root directory (SG-Support). For example, if the SG-Support files are in the following directory:

/usr/spectrum/SG-Support, set SPECROOT to /usr/spectrum.

show children: <relation>: invalid relation

Reason:

The show children command was entered with an invalid relation.

Action:

Re-enter the show children command with a valid relation.

show events: no event format information available

Reason:

The show events command was entered with the -x option, and the CA Spectrum event files containing the event format text messages are not available.

Action:

For the show events command to work with the -x option, which displays probable cause messages for alarms and expanded event messages, OneClick must be installed on the local server, and the environment variable SPECROOT must be set to the path of the Support root directory (SG-Support). For example, if the SG-Support files are in the following directory:

/usr/spectrum/SG-Support, set SPECROOT to /usr/spectrum

show parents: <relation>: invalid relation

Reason:

The show parents command was entered with an invalid relation.

Action:

Re-enter the show parents command with a valid relation.

show rules: <relation>: invalid relation

Reason:

The show rules command was entered with an invalid relation.

Action:

Re-enter the show rules command with a valid relation.

show inheritance: <model_type_handle>: invalid model type handle

Reason:

The show inheritance command was entered with an invalid model_type_handle.

Action:

Re-enter the show inheritance command with a valid model_type_handle.

stopShd: VnmShd not running

Reason:

An attempt was made to run stopShd when the CLI Local Server was not running.

Action:

Start the CLI Local Server.

stopShd: failed

Reason:

The stopShd command failed.

Action:

Try connecting again, and then execute stopShd. If this does not work, kill the VnmShd process manually.

update: <attribute_id>: Attribute not writable

Reason:

No update occurred because an attempt was made to update model attributes that are non-writable.

Action:

Verify that the model attributes you want to update are writable and then re-enter the update command.

update: <attribute_id>: invalid attribute id

Reason:

No update occurred because the update command was entered with an invalid attribute_id.

Action:

Re-enter the update command with a valid attribute_id.

update: <attribute_id>: non shared attribute

Reason:

The update command was used for a model type and an attribute_id of a non-shared attribute was entered.

Action:

Re-enter the update command for the model type, this time using an attribute_id for a shared attribute.

update: <instance_id>: invalid instance id

Reason:

No update occurred because the update command was entered with an invalid instance_id.

Action:

Re-enter the update command with a valid instance_id.

update: <landscape_handle>: invalid landscape handle

Reason:

No update occurred because the update command was entered with an invalid landscape_handle.

Action:

Re-enter the update command with a valid landscape_handle.

update: <model_handle>: invalid model handle

Reason:

No update occurred since the update command was entered with an invalid model_handle.

Action:

Re-enter the update command with a valid model_handle.

update: <model_type_handle>: invalid model type handle

Reason:

No update occurred because the update command was entered with an invalid model_type_handle.

Action:

Re-enter the update command with a valid model_type_handle.

update: <value>: invalid value

Reason:

No update occurred because the update command was entered with an invalid value or values.

Action:

Re-enter the update command with valid values.

update: <action_code>: invalid action code

Reason:

No update occurred because the update command was entered with an invalid action_code.

Action:

Re-enter the update with a valid action_code.

VnmShd: Error: Failed to connect to SpectroSERVER

Reason:

The CLI Local Server failed to connect to the SpectroSERVER.

Action:

Verify that the SpectroSERVER is running.

VnmShd: Error: Lost connection with SpectroSERVER

Reason:

The CLI Local Server, detecting that the SpectroSERVER to which it was connected has terminated, terminates.

Action:

Restart the SpectroSERVER and then run the CLI Local Server.

Appendix C: UNIX to DOS Conversion

On the UNIX platform, CLI commands are typically used with UNIX commands in a terminal window. On the Windows platform, you can use CLI commands with DOS commands in a native DOS window. This appendix lists commonly used UNIX commands and their DOS equivalents.

Note: The appendix is intended to be a quick reference of UNIX and DOS commands rather than an exhaustive list. See your UNIX, DOS, or Windows documentation for more information about commands and their functions.

UNIX	DOS
#	rem
cat	type
cd	cd
chdir	chdir
clear	cls
cmp, diff	comp, fc
cp	copy
cp -r	xcopy
cpio, dump, tar, ufsdump	cpio, dump, tar, ufsdump
cpio, restore, tar, ufsrestore	restore
csh, sh	command
date	date, time
echo	echo
ed	edlin
exit	exit
exportfs, share	share
fdformat, format	format
format	fdisk
format->analyze	scandisk
fsck	chkdsk
goto (csh)	goto

UNIX	DOS
grep	find
if	if
ln -s	subst
lp, lpr	print
ls	dir
ls -l	attrib
man	help
mkdir	md, mkdir
more	more
mv	move, ren, rename
print (sh)	echo
rm	del, erase
rm -r	deltree
rmdir	rd, rmdir
set path= (csh), PATH= (sh)	path
set prompt= (csh), PS1= (sh)	prompt
set var= (csh), var= (sh)	set
shift	shift
showrev	ver
sort	sort
stty	mode
textedit, vi	edit
uncompress, unpack	expand

Index

M

N

O

R

S

T

U

V

W

X