System Management Interface Tool (SMIT)

November 30, 2000

Susan Segura

IBM

System Management Interface Tool (SMIT)

System Management Interface Tool (SMIT)

Contents

System Management Interface Tool (SMIT)

System Management Interface Tool (SMIT)

Introduction

The AIX System Management Interface Tool (SMIT) provides an alternative to the typical method of using complex command syntax, valid parameter values, and custom shell path names for managing and maintaining your operating system configuration.

SMIT offers the following features:

- Two modes of operation
- An interactive, menu-driven user interface
- User assistance
- System management activity logging
- Fast paths to system management tasks
- User-added SMIT screens

For detailed information about the AIX operating system, refer to the following Web address: http://www.ibm.com/servers/aix/library/.

AIX library information is listed under *Technical Publications*.

Modes of Operation

SMIT runs in two modes: ASCII (non-graphical) and Xwindows (graphical). ASCII SMIT can run on both terminals and graphical displays. The graphical mode, which supports a mouse and point-and-click operations, can be run only on a graphical display and with Xwindows support. The ASCII mode is often the preferred way to run SMIT because it can be run from any machine. To start the ASCII mode, type at the command line:

`smitty` or `smit -a`

To start the graphical mode, type:

`smit` or `smit -m`

Note: If you execute the above commands from a terminal or your **TERM** attribute is set to a non-graphical setting, SMIT will always run in the ASCII mode.

End User Interface

SMIT is an interactive, menu-driven user interface that allows you to more easily perform routine system management tasks and to manage and maintain your operating system configuration. System management tasks are grouped by application and presented in a series of menu, selector, and dialog screens. For example, all common software installation tasks are grouped in the Software Installation and Management application. This task-oriented structure makes SMIT easy to use, allowing even novice users to perform routine system administration tasks.

SMIT screens display the actual system configuration. The displayed information varies from system to system, based on what is installed on a particular system. Adding customized system management tasks for your own applications or changing the existing SMIT screen information is one example of what causes this variation. Another example can be seen in the Devices screens. The available system management tasks are based on what type of devices, such as network and storage adapters, disk drives, and other I/O devices, are installed on the system.

SMIT Screens

SMIT uses three types of screens: menu, selector, and dialog screens. SMIT uses the data provided in these screens as options and arguments to create and run high-level command strings to perform a selected task. This data is described in stanza files that are stored in the Object Data Manager (ODM). When you press the Enter key or otherwise start a task from SMIT, the dialog executes a shell script that processes the underlying commands to perform the task. In the SMIT graphical mode, the command string associated with the task displays at the top of the screen as it runs. In the ASCII mode, you can see the command string that will be used before you actually run the task by pressing the F6 Command key.

Menu screens display a list of items that you can select. Menu items are typically system management tasks or classes of tasks that you can perform. Starting from the System Management menu (the main SMIT menu), you select an item defining a broad range of system tasks. You continue to make selections from menus until you reach the final dialog, which typically collects the information and performs the task.

Selector screens, often presented as a pop-up menu, display a list of items from which you specify or select a particular item. Items in a selector screen are typically system objects, such as printers, or the attributes of objects, such as serial or parallel printer mode. The menu screen provides necessary information that is used by the dialog screen.

Dialog screens are the interface to a command or task that you perform. Each dialog executes one or more commands or shell functions. A command can be run from any number of dialogs.

System Management Tasks

You can perform most system management tasks from the SMIT interface. The following table lists the main tasks that display in the System Management menu. Selecting a task from this menu presents additional menus containing tasks, many of which are listed here, that you can perform from that menu.

Application	**System Management Tasks**
Software Installation and Maintenance	Installing new software, updating software, installing fixes, listing installed software, and backing up and restoring the system image.
Software License Management	Adding and deleting node-locked licenses, adding and removing server licenses, managing licenses, and listing licenses.
Devices	Adding, changing, showing, and deleting physical and logical devices; configuring and unconfiguring devices; listing installed devices; and managing PCI hot plugs.

Application	**System Management Tasks**
System Storage Management (Physical & Logical Storage)	Managing logical volumes, volume groups, physical disk drives, and paging space; managing file systems; managing files and directories; and tasks for backing up and restoring the system.
Security and Users	Managing user accounts and groups, passwords, login controls, and roles.
Communications Applications and Services	Configuring all installed communications options and applications, including TCP/IP; NFS server or client; Network Information System (NIS); and Domain Name Service (DNS).
Print Spooling	Configuring and managing printers, print queues, print jobs, and virtual printers.
Problem Determination:	Running hardware diagnostics, performing system traces, initiating system dumps, printing error logs, and verifying software installation and requisites.
Performance and Resource Scheduling	Scheduling jobs, managing resource processes, configuring and enabling Power Management, configuring and using the Workload Manager, running system traces, and reporting system activity.
System Environments	Starting and stopping the system; configuring and managing system environment parameters such as language, date, user interface, and time; managing system logs; managing the remote reboot facility; and managing system hang detection.
Processes and Subsystems	Managing subsystems, processes, and subservers.

One other menu item, **Applications**, is provided in the System Management menu so that you can add your own dialog and menu screens to support other applications.

Object Data Manager (ODM)

The Object Data Manager (ODM) stores information about the system in a binary database. This information is stored as objects, with their attributes and associated characteristics, and managed by the ODM. SMIT objects that the ODM manages include display information for dialog, menu, and selector screens. When SMIT runs commands to perform a task, the commands retrieve information from the ODM.

The SMIT Database

SMIT objects are generated with ODM creation facilities and stored in files in a designated database. The default SMIT database consists of the following eight files:

- sm_menu_opt
- sm_menu_opt.vc
- sm_name_hdr
- sm_name_hdr.vc

- sm_cmd_hdr
- sm_cmd_hdr.vc
- sm_cmd_opt
- sm_cmd_opt.vc

These files are usually stored in the **/usr/lib/objrepos** directory. They should always be saved and restored together.

User Assistance

User assistance is provided for menus, menu choices, and input and output fields. In the SMIT ASCII mode, press the F1 Help key to display context-sensitive help. In the graphical mode, select the desired help from the Help menu.

System Management Activity Logging

SMIT logs all system management activity in two files. These files usually reside in the user's home directory. The **smit.log** file records all SMIT actions, such as the name of each screen you display, the command string it ran, the output from the command string, and any error output. The **smit.script** file records all high-level command strings that the system executes. All entries in these two files are date stamped.

Fast Paths

You can use fast paths for virtually all of the tasks that you run from SMIT. Fast paths are command strings that, when executed with the SMIT command, bypass dialog and menu screens and go directly to the menu or dialog screen from which you can perform a specific task. Many of the fast paths are the same commands that are run from the SMIT screens. Any number of fast paths can point to the same menu, selector, or dialog screen. To invoke a fast path, type the command to start SMIT followed by the fast path. For example:

```
smitty dev
```

starts SMIT in the ASCII mode, bypasses the System Management main menu, and takes you directly to the Devices menu. To invoke the same fast path in the SMIT graphical mode, you would type `smit dev`. In the ASCII mode, you can see the fast path for the current screen by pressing the F8 Image key. To see fast paths in SMIT's graphical mode, select **Fast path** from the Show menu.

The fast paths for the tasks in the SMIT System Management menu are:

Application	Fast Path
Software Installation and Maintenance	**install**
Software License Management	**licenses**
Devices	**dev**
System Storage Management (Physical & Logical Storage)	**storage**
Security and Users	**security**
Communications Applications and Services	**commo**
Print Spooling	**spooler**
Problem Determination	**problem**
Performance and Resource Scheduling	**performance**

Application	Fast Path
System Environments	**system**
Processes and Subsystems	**src**

Fast paths are also available for most of the other system management tasks that belong to subsequent SMIT menus. See "Appendix C: Fast Paths for SMIT Tasks" on page 21 for a list of additional fast paths.

When you add menu and dialog screens to support your own installed applications, you can generate fast paths for them and for the system management tasks in these screens. See "Defining Fast Paths" on page 12 for more information.

Adding Dialog and Menu Screens for Customer Applications

You can build your own menu, selector, and dialog screens to support the system management tasks in your own installed applications and add them to the SMIT database. This procedure involves the following steps:

- Designing and creating SMIT screens
- Creating stanza files
- Creating a test database
- Testing the stanza files
- Adding the stanza files to the SMIT database

Before you start, it is helpful to understand the purpose of each of these screens, what to consider in designing them, and how they are built.

SMIT Menu Screens

A menu is the basic entry point into SMIT and can be followed by another menu, selector, or dialog screen. Menus present a list of tasks. Selecting a task from one menu can lead to another menu or to a selector or dialog screen. The following example of a menu shows the Users menu from the SMIT Security & Users application:

```
                    Users

Move cursor to desired item and press Enter.

  Add a User
  Change a User's Password
  Change / Show Characteristics of a User
  Lock / Unlock a User's Account
  Reset User's Failed Login Count
  Remove a User
  List All Users


F1=Help         F2=Refresh     F3=Cancel     Esc+8=Image
Esc+9=Shell     Esc+0=Exit     Enter=Do
```

Design menus to help the SMIT user narrow the scope of choice to a particular task. Your design can be as simple as a new menu and dialog attached to an existing SMIT menu, or as complex as an entire new hierarchy of menus, selectors, and dialogs that start at the SMIT Applications menu.

You build menus by defining them in a stanza file. You can define any number of menus in one or more stanza files, along with selector and dialog screens. Menus consist of objects that are instances of object classes. The object class used in menus is **sm_menu_opt**. A typical menu contains one or more objects, each with its own unique ID, that is a member of the **sm_menu_opt** object class. For example, a menu with two items uses the object class and a unique ID to identify the title of the screen, another for the first item in the menu, and another for the second item in the menu.

When an option is selected from a menu screen, SMIT collects all menu objects with the same ID from the object repository, then builds a screen that is presented to the user. This process is repeated with each successive menu that the user visits. To add a new item to a SMIT menu, you must define a menu object that uses the same ID as the other objects in that menu.

SMIT Selector Screens

Selector screens are used to obtain information that subsequent screens need or to select the selector or dialog screen to use next. Selector screens usually prompt the user for input in a response area or to select a value from a pop-up list. Typically, a question field displays and the user types or selects a value from a list or option ring in the response area. The following examples show how a selector is used.

Selecting **Change a User's Password** from the Users menu below, displays a selector screen.

```
                     Users

Move cursor to desired item and press Enter.

  Add a User
  Change a User's Password
  Change / Show Characteristics of a User
  Lock / Unlock a User's Account
  Reset User's Failed Login Count
  Remove a User
  List All Users


F1=Help         F2=Refresh      F3=Cancel     F8=Image
F9=Shell        F10=Exit        Enter=Do
```

This is the selector screen:

```
         Change / Show Characteristics of a User

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                           [Entry Fields]
* User NAME                                []            +




F1=Help         F2=Refresh      F3=Cancel      F4=List
Esc+5=Reset     F6=Command      F7=Edit        F8=Image
F9=Shell        F10=Exit        Enter=Do
```

Design selector screens to request only one piece of information from the user. For example, the name of a user. You can string selector screens together in a series to gather several pieces of information before a dialog displays. For example, the name, user ID, and password for a user.

You build selectors by defining them in a stanza file. You can define any number of selectors in one or more stanza files, along with menu and dialog screens. Selectors consist of objects that are instances of object classes. The object classes used in selectors are **sm_name_hdr**, typically used for identifying the title of the selector screen or other attributes, and **sm_cmd_hdr**, which is used for an entry field or pop-up list.

If you want to provide a pop-up list of choices, associate the selector **sm_cmd_opt** object class with a **cmd_to_list** descriptor that lists the valid choices. The list is not hard-coded, but developed by the command together with standard output. You get this list by selecting the **F4 List** key in a SMIT screen.

If you want a pop-up list to display, but not the selector screen, define a ghost selector, using the **ghost=″y″** descriptor of the **sm_cmd_hdr** object class.

A super-ghost selector permits branching after a menu selection, where the branch to be taken depends on the system state and not user input. In this case, you can use the **cmd_to_classify** descriptor in the super-ghost selector to get the required information and select the correct screen to display next.

SMIT Dialog Screens

A dialog screen is the final panel in a SMIT sequence. This screen is where any remaining user input is requested and where the selected task is actually run. Shown below is an example of a a dialog screen.

```
                       Add a Group

Type or select values in entry fields.
Press Enter AFTER making all desired changes.
                                            [Entry Fields]
* Group NAME                              []
  ADMINISTRATIVE group?                   false         +
  Group ID                                []             #
  USER list                               []            +
  ADMINISTRATOR list                      []            +

F1=Help        F2=Refresh       F3=Cancel      F4=List
Esc+5=Reset    F6=Command       F7=Edit        F8=Image
F9=Shell       F10=Exit         Enter=Do
```

To design a dialog, you must know the command string that you want to build and the command options and operands for any values that the user can specify. In the dialog screen, these command options and operands display in ″user-oriented″ language to prompt the user for a response or selection.

To build a dialog, you must first define it in a stanza file. You can define several dialogs in a single stanza file, along with menu and selector screens. The object classes used in dialog screens are **sm_cmd_hdr**, which is used for the title of your screen and command string, and **sm_cmd_opt**, which is used for each entry field in the dialog.

To provide a run-time list of choices, associate each dialog object with a command that lists the valid choices. These commands are defined in the **cmd_to_list** field. This list displays when you select the **F4 List** key in a SMIT screen. Selecting this key causes SMIT to run the command defined in the associated **cmd_to_list** field and to use its standard output and **stderr** file to develop the list. All the values are typically obtained from the preceding selector screens.

In a ghost dialog, the dialog screen does not display. The dialog runs as if you had immediately pressed the dialog screen's **Enter** key to run the dialog.

Designing and Creating SMIT Screens

Use the following procedure as a guideline for designing and creating your own menu, selector, and dialog screens. Adding your own applications may require more steps than are described here. For a more detailed explanation and examples of SMIT screen types and object classes, refer to *General Programming Concepts: Writing and Debugging Programs* in the AIX Documentation Library Service. You can find the library service at:

```
http://www.ibm.com/servers/aix/library
```

AIX library information is listed under *Technical Publications*.

1. Determine where in the existing SMIT screens you want to add one or more menu, selector, and dialog screens for your application. One way to do this is to start SMIT and look through the various screens to find any that perform tasks similar to those you want to add. Even if you prefer to add your entry menu to the Applications menu, which is provided for that purpose, going through the various screens will assist you in designing your screens. To start SMIT, type:

   `smitty` or `smit`
2. Look through the SMIT screens and, when you have decided where to add your menus, dialog, and optional selector screens, exit from SMIT.
3. Either remove the **smit.log** file and start SMIT again, or restart SMIT using the following syntax (replacing *my_smit.log* with a file name you choose):

   ```
   smitty -t -l my_smit.log
   ```

   The **-l** flag redirects output to a log file other than **smit.log** when you start SMIT. The **-t** flag records detailed trace information in the designated log file. Using these flags allows you to isolate the trace output of this session. If you prefer, you can remove the **smit.log** file (usually located in the home directory) instead of redirecting output to another file.
4. From the System Management menu, select the desired application, then go through the sequence of menu screens until you get to the menu to which you want to add the entry menu for your application. Continue going through any subsequent menus until you get to the final dialog screen. As you do this, the object class IDs and other information for each of the screens you access are logged in the current SMIT log file. You will need these object class IDs to create the stanza file for your menu, selector, and dialog screens. Do this step even if you are adding your application to the Applications menu.
5. Using an ASCII editor or the **pg** command, open the SMIT log file you specified above to find the IDs for the object classes defined in each menu.

   ```
   pg my_smit.log
   ```

   The following example, from the Security & Users application, shows some of the information that is logged when you go through the menus to get to the menu where you add a user:

```
Object class: sm_menu_opt,
   id   = "__ROOT__", id_seq_num = "0",
   next_id = "top_menu",
   text = "System Management"

(Menu screen selected,
   FastPath    = "top_menu",
   id_seq_num  = "0",
   next_id     = "top_menu",
   title       = "System Management".)

Object class: sm_menu_opt,
   id   = "top_menu", id_seq_num = "010",
   next_id = "install",
   text = "Software Installation and Maintenance"

Object class: sm_menu_opt,
   id   = "top_menu", id_seq_num = "015",
   next_id = "licenses",
   text = "Software License Management"

Object class: sm_menu_opt,
   id   = "top_menu", id_seq_num = "020",
   next_id = "dev",
   text = "Devices"

Object class: sm_menu_opt,
   id   = "top_menu", id_seq_num = "030",
   next_id = "storage",
   text = "System Storage Management
   (Physical & Logical Storage)"

Object class: sm_menu_opt,
   id   = "top_menu", id_seq_num = "100",
   next_id = "apps",
   text = "Applications"

Object class: sm_menu_opt,
   id   = "top_menu", id_seq_num = "999",
   next_id = "",
   text = "Using SMIT (information only)"
.
.
.
Object class: sm_menu_opt,
   id   = "top_menu", id_seq_num = "040",
   next_id = "security",
   text = "Security & Users"

(Menu screen selected,
   FastPath    = "security",
   id_seq_num  = "040",
   next_id     = "security",
   title       = "Security & Users".)

Object class: sm_menu_opt,
   id   = "security",
   id_seq_num = "010",
   next_id = "users",
   text = "Users"

(Menu screen selected,
   FastPath    = "users",
   id_seq_num  = "010",
   next_id     = "users",
   title       = "Users".)
```

```
Object class: sm_cmd_hdr,
   id   = "mkuser",
   option_id = "user_add",
   name = "Add a User"

(Dialogue screen selected,
   FastPath    = "mkuser",
   id          = "mkuser",
   title       = "Add a User".)

Object class: sm_cmd_opt,
   id   = "user_add",
   id_seq_num = "01",
   name = "User NAME"

Object class: sm_cmd_opt,
   id   = "user_add",
   id_seq_num = "02",
   name = "User ID"

Object class: sm_cmd_opt,
   id   = "user_add",
   id_seq_num = "03",
   name = "ADMINISTRATIVE USER?"
.
.
.
```

6. Record all the object class IDs that you need for defining your SMIT screens in the stanza files you will be creating. For example, if you want to add another option to the Users menu, the **sm_menu_opt** object class ID that you need is **id = ″security″** as shown in the example below:

```
Object class: sm_menu_opt,
   id   = "security",
   id_seq_num = "010",
   next_id = "users",
   text = "Users"

(Menu screen selected,
   FastPath    = "users",
   id_seq_num  = "010",
   next_id     = "users",
   title       = "Users".)
```

   From this same output, you can determine the object class IDs and commands used in the dialog screen for the task of adding a user, shown below:

```
Object class: sm_cmd_hdr,
   id   = "mkuser",
   option_id = "user_add",
   name = "Add a User"

(Dialogue screen selected,
   FastPath    = "mkuser",
   id          = "mkuser",
   title       = "Add a User".)
```

You are now ready to create your stanza files. For more information, see “Creating Stanza Files”.

Creating Stanza Files

You can use existing stanza files to create new stanza files that define and build menu, selector, and dialog screens for your applications. After you create your stanza files, you add them to a test database, test them, then add them to the SMIT database.

Use the following procedure as a guideline for creating your stanza files. You can also use, if you prefer, the demo application in "Appendix B: SMIT Example Programs" on page 14.

1. Set the **ODMDIR** environment variable:

   ```
   export ODMDIR=/usr/lib/objrepos
   ```

   The **/usr/lib/objrepos** directory is the default object repository for system information and can be used to store your compiled objects. At SMIT run time, the objects are automatically retrieved from a SMIT database.
2. Use the **odmget** command with the object class IDs you previously recorded to retrieve the existing stanza files. For example, if you want to find the stanza file for the Security and Users menu, type the following:

   ```
   odmget -q id=security sm_menu_opt
   ```

   In the displayed output, you find the following stanza file:

   ```
   sm_menu_opt:
           id_seq_num = "010"
           id = "security"
           next_id = "users"
           text = "Users"
           text_msg_file = "smit.cat"
           text_msg_set = 25
           text_msg_id = 35
           next_type = "m"
           alias = ""
           help_msg_id = "3004100"
           help_msg_loc = ""
           help_msg_base = ""
           help_msg_book = ""
   ```

   To find stanza files for the Users menu, type:

   ```
   odmget -q id=users sm_menu_opt | pg
   ```

   The output includes the following stanza file:

   ```
   sm_menu_opt:
           id_seq_num = "010"
           id = "users"
           next_id = "mkuser"
           text = "Add a User"
           text_msg_file = "smit.cat"
           text_msg_set = 25
           text_msg_id = 166
           next_type = "d"
           alias = ""
           help_msg_id = "1800168"
           help_msg_loc = ""
           help_msg_base = ""
           help_msg_book = ""
   ```

   To find stanza files for the Add User dialog, type:

   ```
   odmget -q id=mkuser sm_cmd_hdr | pg
   ```

   The output includes the following stanza file:

   ```
   sm_cmd_hdr:
           id = "mkuser"
           option_id = "user_add"
           has_name_select = "n"
           name = "Add a User"
           name_msg_file = "smit.ca
           name_msg_set = 25
   ```

```
        name_msg_id = 166
        cmd_to_exec = "x() {\n\
LIST=\n\
SET_A=\n\
SET_A=\n\
for i in \"$@\"\n\
do\n\
        if [ \"$i\" = \"admin=true\" ]
        then\n\
                SET_A=\"-a\"\n\
                continue\n\
        fi\n\
        LIST=\"$LIST \\\"$i\\\"\"\n\
done\n\
eval mkuser $SET_A $LIST\n\
}\n\
x"
        ask = "n"
        exec_mode = ""
        ghost = "n"
        cmd_to_discover = "lsuser -D"
        cmd_to_discover_postfix = ""
        name_size = 0
        value_size = 0
        help_msg_id = "1800168"
        help_msg_loc = ""
        help_msg_base = ""
        help_msg_book = ""
```

3. Look in the SMIT log file for the command strings used when the screens are run to see if special tools are being used (such as **sed** or **awk** scripts, **ksh** shell functions, environment variable assignment, and other tools).

   Command strings are processed twice: the first time by the **odmadd** command, and the second time by the **ksh** shell. Be careful when using special escape metacharacters such as \ or quotation characters (' and ″). Using these characters incorrectly can alter the meaning of commands and prevent them from executing. Notice that the output of the **odmget** command does not always match the input to the **odmadd** command, especially when special characters or multiple-line string values are used.
4. Copy the stanza files found in the above steps to define your new menu, selector, and dialog objects and to create new stanza files.

For detailed information, and to see examples of stanzas used to code SMIT objects, refer to the *SMIT Example Program* in *General Programming Concepts: Writing and Debugging Programs* in the AIX Documentation Library Service. You can find the library service at:

`http://www.ibm.com/servers/aix/library`

AIX library information is listed under *Technical Publications*.

Defining Fast Paths

You can use a SMIT **sm_menu_opt** object to define a fast path for new system administration tasks that, used together with the **smit** command, takes you directly to a specific menu, selector, or dialog. The alias you created does not display.

To build aliases and fast paths, define them in a stanza file. You can define several menus, selectors, and dialogs in a single file. An **sm_menu_opt** object defines a fast path by setting the **alias** field to ″y″. The new fast path or alias name is specified by the value in the **id** field. The contents of the **next_id** field point to another menu object, selector header object, or dialog header object, depending on whether the value of the **next_type** field is *m* (menu), *n* (selector), or *d* (dialog).

Every **sm_menu_opt** object that is not an alias for a menu title (`next_type="m"`) should have a unique **next_id** field value, since this field is automatically used as a fast path. If you want two menu items to point to the same successor menu, one of the **next_id** fields should point to an alias, which in turn points to the successor menu.

Creating a Test Database

It is recommended that you set up a test database when you develop new objects so that you can test new stanza files before adding them to the SMIT database. To create a test database:

1. Create a directory to use for testing. In the following example, a `/home/smit/test` directory is created:
   ```
   mkdir /home/smit/test
   ```
2. Make the test directory the current directory:
   ```
   cd /home/smit/test
   ```
3. To define the test directory as the default object repository, set the **ODMDIR** environment variable to the current directory:
   ```
   export ODMDIR=.
   ```
4. Create a new SMIT database in the test directory:
   ```
   cp /usr/lib/objrepos/sm_* $ODMDIR
   ```

   Note: Always back up the **/usr/lib/objrepos** directory before deleting or adding any objects or object classes. Unanticipated damage to objects or classes needed for system operations can cause system problems.

Testing Stanza Files

1. Use the **odmadd** command and your stanza file name to add your new stanza files to your test database so that you can ensure that they work. For example:
   ```
   odmadd my_stanza_file
   ```

   Replace the file `my_stanza_file` with the name of your stanza file.
2. In your local test database directory, start SMIT so that you can test and debug your additions:
   ```
   smit -o .
   ```
3. When you are finished testing, restore the **/usr/lib/objrepos** directory as the default object repository by setting the **ODMDIR** environment variable to **/usr/lib/objrepos**:
   ```
   export ODMDIR=/usr/lib/objrepos
   ```

Adding Stanza Files to the SMIT Database

Use the **odmadd** command and your stanza file name to add your new stanza files to the SMIT database. For example:

```
odmadd my_stanza_file
```

Replace the file `my_stanza_file` with the name of your stanza file.

Learning More About SMIT

For additional and more detailed information about SMIT, refer to the chapter titled *System Management Interface Tool (SMIT)* in *General Programming Concepts: Writing and Debugging Programs*. You can find this and other related books in the AIX Documentation Library Service located at:

```
http://www.ibm.com/servers/aix/library
```

AIX library information is listed under *Technical Publications*.

Appendix A: Special Notices

Appendix B: SMIT Example Programs

You can use these example programs as models for developing your own stanza files. These are functioning programs that you can add to the SMIT database and access from SMIT by selecting **Applications** in the System Management menu. It is recommended that you add the stanza files to a test database first.

Example Program One

First, decide where to insert the menu for your application. Your new menu will point to other menus, name headers, and dialogs. For this example, the menu is being added under the Applications menu. The **next_id** for the Applications menu item is ″apps″, so a **menu_opt** with the ID ″apps″ is also created.

```
sm_menu_opt:
   id          = "apps"
   id_seq_num  = "010"
   next_id     = "demo"
   text        = "SMIT Demos"
   next_type   = "m"

sm_menu_opt:
   id          = "demo"
   id_seq_num  = "010"
   next_id     = "demo_queue"
   text        = "Demo 1: Add a Print Queue"
   next_type   = "n"

sm_menu_opt:
   id             = "demo"
   id_seq_num     = "020"
   next_id        = "demo_mle_inst_lang_hdr"
   text           = "Demo 2: Add Language for Application Already Installed"
   next_type      = "n"
```

The following text creates an alias for **demo_mle_inst_lang_hdr** so that it is easier to remember.

```
sm_menu_opt:
   id             = "demo_lang"
   next_id        = "demo_mle_inst_lang_hdr"
   next_type      = "n"
   alias          = "y"

sm_menu_opt:
   id_seq_num     = "030"
   id             = "demo"
   next_id        = "demo_lspv"
   text           = "Demo 3: List Contents of a Physical Volume"
   text_msg_file  = "smit.cat"
   next_type      = "n"

sm_menu_opt:
   id_seq_num     = "040"
   id             = "demo"
   next_id        = "demo_date"
   text           = "Demo 4: Change / Show Date, Time"
   text_msg_file  = "smit.cat"
   next_type      = "n"
```

Next, you will create a task called ″Add a print queue″. If the printers.rte package is not installed, it will install it automatically. If the user is running MSMIT (SMIT in a windows interface), it will launch a graphical program for this task. Otherwise, it will branch to the SMIT print queue task.

The following items are used in the example files:

1. cooked output and cmd_to_classify (page 16)
2. (SMIT environment variable (msmit vs. ascii))
3. ghost name_hdr (page 17)
4. super-ghost name_hdr (page 16)
5. creating an ″OK/Cancel″ option (page 17)
6. dspmsg for translations (page 17)
7. exit/exec mode (page 17)
8. id_seq_num for a name_hdr option (page 17)

Item 1 and Item 4:
Note that the **next_id** is the same as the **id**. Remember that the output of the **cmd_to_classify** is appended to the **next_id** because the type is ″c″ (cooked). So, the **next_id** will be either **demo_queue1** or **demo_queue2**. None of the output of the **name_hdr** is displayed, and there is no **cmd_to_list** in the **demo_queue_dummy_opt**, which makes this **name_hdr** a super-ghost.

```
sm_name_hdr:
   id                        = "demo_queue"
   next_id                   = "demo_queue"
   option_id                 = "demo_queue_dummy_opt"
   name                      = "Add a Print Queue"
   name_msg_file             = "smit.cat"
   name_msg_set              = 52
   name_msg_id               = 41
   type                      = "c"
   ghost                     = "y"
   cmd_to_classify         = "\
x()
{
    # Check to see if the printer file is installed.
    lslpp -l printers.rte 2>/dev/null 1>/dev/null
    if [[ $? != 0 ]]
    then
   echo 2
    else
   echo 1
    fi
}
x"
   next_type                  = "n"
```

Items 2 and 4:
Having determined that the printer software is installed, we want to know if the graphical SMIT or the ASCII SMIT should run for this task. To do this, we check the value of the SMIT environment variable, which is ″m″ for windows (Motif) or ″a″ for ASCII. Here, again, we tack the output of the **cmd_to_classify** onto the **next_id**.

```
sm_name_hdr:
   id                        = "demo_queue1"
   next_id                   = "mkpq"
   option_id                 = "demo_queue_dummy_opt"
   has_name_select           = ""
   ghost                     = "y"
   next_type                 = "n"
   type                      = "c"
   cmd_to_classify           = "\
gui_check()
{
   if [ $SMIT = \"m\" ]; then
     echo gui
   fi
}
   gui_check"

sm_name_hdr:
   id              = "mkpqgui"
   next_id         = "invoke_gui"
   next_type       = "d"
   option_id       = "demo_queue_dummy_opt"
   ghost           = "y"
```

Item 7:
Note that the **exec_mode** of this command is "e", which exits SMIT before running the **cmd_to_exec**.

```
sm_cmd_hdr:
   id                = "invoke_gui"
   cmd_to_exec       = "/usr/bin/X11/xprintm"
   exec_mode         = "e"
   ghost             = "y"

sm_cmd_opt:
   id                             = "demo_queue_dummy_opt"
   id_seq_num                     = 0
```

Item 3 and Item 5:
The printer software is not installed. Install the software and loop back to **demo_queue1** to check the SMIT environment variable. This is a **ghost name_hdr**. The **cmd_to_list** of the **sm_cmd_opt** is displayed immediately as a pop-up option instead of waiting for the user to input a response. In this ghost, the **cmd_opt** is a simple OK/Cancel box that prompts the user to press return.

```
sm_name_hdr:
   id                        = "demo_queue2"
   next_id                   = "demo_queue1"
   option_id                 = "demo_queue_opt"
   name                      = "Add a Print Queue"
   name_msg_file             = "smit.cat"
   name_msg_set              = 52
   name_msg_id               = 41
   ghost                     = "y"
   cmd_to_classify           = "\
install_printers ()
{

  # Install the printer package.
  /usr/lib/assist/install_pkg \"printers.rte\" 2>&1 >/dev/null
  if [[ $? != 0 ]]
  then
    echo "Error installing printers.rte"
    exit 1
  else
    exit 0
  fi
}
install_printers "
   next_type                        = "n"
```

Item 5, Item 6, and Item 8:
Here a **cmd_opt** is used as an OK/Cancel box. The **dspmsg** command is used to display the text for the option. This allows for translation of the messages. (The **id_seq_num** for the option is 0. Only one option is allowed per **name_hdr**, and its **id_seq_num** must be 0.)

```
sm_cmd_opt:
   id                        = "demo_queue_opt"
   id_seq_num                = "0"
   disc_field_name           = ""
   name                      = "Add a Print Queue"
   name_msg_file             = "smit.cat"
   name_msg_set              = 52
   name_msg_id               = 41
   op_type                   = "l"
   cmd_to_list               = "x()\
{
if [ $SMIT = \"a\" ] \n\
then \n\
dspmsg -s 52 smit.cat 56 \
'Press Enter to automatically install the printer software.\n\
Press F3 to cancel.\n\
'\n\
else \n\
dspmsg -s 52 smit.cat 57 'Click on this item to automatically install
the printer software.\n' \n\
fi\n\
} \n\
x"
   entry_type              = "t"
   multi_select            = "n"
```

Example Program Two

The goal in this example is to ″Add a Language for an Application Already Installed″. It is often more clear to the user to get some information before displaying the dialog screen. Name Headers (**sm_name_hdr**) can be used for this purpose. In this example, two name headers are used to determine the language to install and the installation device. The dialog has entries for the rest of the information needed perform the task.

The example files in this section show how to:

1. Save output from successive **name_hdrs** with **cooked_field_name** (page 18).
2. Use **getopts** inside **cmd_to_exec** to process **cmd_opt** info (page 19).
3. Use a ring list instead of **cmd_to_list** to display values **cmd_opts** (page 21).

Item 1: Saving output from successive **name_hdrs** with **cooked_field_name**

This is the first **name_hdr**. It is called by the **menu_opt** for this function. We want to save the user's input for later use in the dialog. The parameter passed into the **cmd_to_classify** comes from the user's selection or entry. **Cmd_to_classify** cleans up the output and stores it in the variable specified by **cooked_field_name**. This overrides the default value for the **cmd_to_classify** output, which is **cookedname**. The default must be overridden because we also need to save the output of the next **name_hdr**.

```
sm_name_hdr:
    id                          = "demo_mle_inst_lang_hdr"
    next_id                     = "demo_mle_inst_lang"
    option_id                   = "demo_mle_inst_lang_select"
    name                        = "Add Language for Application Already Installed"
    name_msg_file               = "smit.cat"
    name_msg_set                = 53
    name_msg_id                 = 35
    type                        = "j"
    ghost                       = "n"
    cmd_to_classify             = "\
        foo() {
            echo $1 | sed -n \"s/[ˆ[]*\\[\\([ˆ]]*\\).*/\\1/p\"
        }
        foo"
    cooked_field_name           = "add_lang_language"
    next_type                   = "n"
    help_msg_id                 = "2850325"

sm_cmd_opt:
    id                          = "demo_mle_inst_lang_select"
    id_seq_num                  = "0"
    disc_field_name             = "add_lang_language"
    name                        = "LANGUAGE translation to install"
    name_msg_file               = "smit.cat"
    name_msg_set             = 53
    name_msg_id              = 20
    op_type                  = "l"
    entry_type               = "n"
    entry_size               = 0
    required                 = ""
    prefix                   = "-l "
    cmd_to_list_mode         = "a"
    cmd_to_list              = "/usr/lib/nls/lsmle -l"
    help_msg_id              = "2850328"
```

This is the second **name_hdr**. Here the user's input is passed directly through the **cmd_to_classify** and stored in the **variable add_lang_input**.

```
sm_name_hdr:
    id                       = "demo_mle_inst_lang"
    next_id                  = "demo_dialog_add_lang"
    option_id                = "demo_add_input_select"
    has_name_select          = "y"
    name                     = "Add Language for Application Already Installed"
    name_msg_file            = "smit.cat"
    name_msg_set             = 53
    name_msg_id              = 35
    type                     = "j"
    ghost                    = "n"
    cmd_to_classify          = "\
        foo() {
            echo $1
        }
        foo"
    cooked_field_name        = "add_lang_input"
    next_type                = "d"
    help_msg_id              = "2850328"

sm_cmd_opt:
    id                       = "demo_add_input_select"
    id_seq_num               = "0"
    disc_field_name          = "add_lang_input"
    name                     = "INPUT device/directory for software"
    name_msg_file            = "smit.cat"
    name_msg_set             = 53
    name_msg_id              = 11
    op_type                  = "l"
    entry_type               = "t"
    entry_size               = 0
    required                 = "y"
    prefix                   = "-d "
    cmd_to_list_mode         = "1"
    cmd_to_list              = "/usr/lib/instl/sm_inst list_devices"
    help_msg_id              = "2850313"
```

Item 2: Using **getopts** inside **cmd_to_exec** to process **cmd_opt** info

Each of the **cmd_opts** formats its information for processing by the **getopts** command (a dash and a single character, followed by an optional parameter). The colon following the letter in the **getopts** command means that a parameter is expected after the dash option. This is a nice way to process the **cmd_opt** information if there are several options, especially if one of the options could be left out, causing the sequence of $1, $2, etc. to get out of order.

```
sm_cmd_hdr:
    id             = "demo_dialog_add_lang"
    option_id      = "demo_mle_add_app_lang"
    has_name_select = ""
    name      = "Add Language for Application  Already Installed"
    name_msg_file   = "smit.cat"
    name_msg_set    = 53
    name_msg_id     = 35
    cmd_to_exec     = "\
       foo()
       {
       while getopts d:l:S:X Option \"$@\"
       do
          case $Option in
             d) device=$OPTARG;;
             l) language=$OPTARG;;
             S) software=$OPTARG;;
             X) extend_fs="-X";;
          esac
       done

       if [[ ′/usr/lib/assist/check_cd -d $device′ = '1' ]]
       then
          /usr/lib/assist/mount_cd $device
          CD_MOUNTED=true
       fi

       if [[ $software = \"ALL\" ]]
       then
          echo "Installing all software for $language..."
       else
          echo "Installing $software for $language..."
       fi
       exit $RC
       }
       foo"
    ask                       = "y"
    ghost                     = "n"
    help_msg_id               = "2850325"

sm_cmd_opt:
    id                        = "demo_mle_add_app_lang"
    id_seq_num                = "0"
    disc_field_name           = "add_lang_language"
    name                      = "LANGUAGE translation to install"
    name_msg_file             = "smit.cat"
    name_msg_set              = 53
    name_msg_id               = 20
    entry_type                = "n"
    entry_size                = 0
    required                  = "y"
    prefix                    = "-l "
    cmd_to_list_mode          = "a"
    help_msg_id               = "2850328"
```

The prefix field precedes the value selected by the user, and both the prefix and the user-selected value are passed into the **cmd_to_exec** for **getopts** processing.

```
sm_cmd_opt:
    id                    = "demo_mle_add_app_lang"
    id_seq_num            = "020"
    disc_field_name       = "add_lang_input"
    name                  = "INPUT device/directory for software"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id           = 11
    entry_type            = "n"
    entry_size            = 0
    required              = "y"
    prefix                = "-d "
    cmd_to_list_mode      = "1"
    cmd_to_list           = "/usr/lib/instl/sm_inst list_devices"
    help_msg_id           = "2850313"
```

```
sm_cmd_opt:
    id                   = "demo_mle_add_app_lang"
    id_seq_num           = "030"
    name                 = "Installed APPLICATION"
    name_msg_file        = "smit.cat"
    name_msg_set         = 53
    name_msg_id          = 43
    op_type              = "l"
    entry_type           = "n"
    entry_size           = 0
    required             = "y"
    prefix               = "-S "
    cmd_to_list_mode     = ""
    cmd_to_list          = "\
       list_messages ()
       {
          language=$1
          device=$2
          lslpp -Lqc | cut -f2,3 -d':'
       }
       list_messages"
    cmd_to_list_postfix   = "add_lang_language add_lang_input"
    multi_select         = ","
    value_index          = 0
    disp_values          = "ALL"
    help_msg_id          = "2850329"
```

Item 3: Using a ring list instead of **cmd_to_list** to display values **cmd_opts**

Here, instead of a **cmd_to_list**, there is a set of Ring values delimited by a comma in the **disp_values** field. This list displays one item at a time as the user presses a tab in the **cmd_opt** entry field. However, instead of passing a ″yes″ or ″no″ to the **cmd_hdr**, it is more useful to use the **aix_values** field to pass either a ″-X″ or nothing. The list in the **aix_values** field must match one-to-one with the list in the **disp_values** field.

```
sm_cmd_opt:
   id_seq_num = "40"
   id = "demo_mle_add_app_lang"
   disc_field_name = ""
   name = "EXTEND file systems if space needed?"
   name_msg_file = "smit.cat"
   name_msg_set = 53
   name_msg_id = 12
   op_type = "r"
   entry_type = "n"
   entry_size = 0
   required = "y"
   multi_select = "n"
   value_index = 0
   disp_values = "yes,no"
        values_msg_file = "sm_inst.cat"
   values_msg_set = 1
   values_msg_id = 51
   aix_values = "-X,"
   help_msg_id = "0503005"
```

Appendix C: Fast Paths for SMIT Tasks

This section contains fast paths for many of the tasks you can perform from SMIT. To invoke a fast path, type the command to start SMIT with the desired fast path command. For example:

```
smitty dev
```

System Management Menu

Fast path to menu: **top_menu**

The System Management Menu is the SMIT main menu. The following table lists the fast paths to the main SMIT application menus:

Application	Fast Path
Software Installation and Maintenance	**install**
Software License Management	**licenses**
Devices	**dev**
System Storage Management (Physical & Logical Storage)	**storage**
Security and Users	**security**
Communications Applications and Services	**commo**
Print Spooling	**spooler**
Problem Determination	**problem**
Performance and Resource Scheduling	**performance**
System Environments	**system**
Processes and Subsystems	**src**

Software Installation and Maintenance

Fast path to menu: **install**

Task	Fast Path
Install and Update Software Install Software Update Installed Software to Latest Level (Update All) Install Software Bundle Update Software by Fix (APAR) Install and Update from ALL Available Software	**install_update** **install_latest** **update_all** **install_bundle** **update_by_fix** **install_all**
List Software and Related Information List Installed Software and Related Information List Installed Software List Applied but Not Committed Software Updates Show Software Installation History Show Fix (APAR) Installation Status List Fileset Requisites List Fileset Dependents List Files Included in a Fileset List Fileset Containing File Show Installed License Agreements	**list_software** **list_installed** **list_installed_sw** **list_applied_sw** **show_history** **show_apar_stat** **list_requisites** **list_dependents** **list_files** **what_fileset** **installed_license**

Task	Fast Path
List Software on Media and Related Information List Filesets in a Bundle List Software on Installation Media List Software Fixes (APARs) on Installation Media List Supplemental Fileset Information on Installation Media Show License Agreements on Installation Media	**list_media** **list_bundle** **list_media_sw** **list_media_fixes** **list_media_info** **license_on_media**
Software Maintenance and Utilities Commit Applied Software Updates (Remove Saved Files) Reject Applied Software Updates (Use Previous Version) Remove Installed Software Copy Software to Hard Disk for Future Installation Check Software File Sizes After Installation Verify Software Installation and Requisites	**maintain_software** **commit** **reject** **remove** **bffcreate** **check_files** **verify_install**
Network Installation Management Configure Network Installation Management Client Fileset Install and Update Software List Software on Media and Related Information List Filesets in a Bundle List Software on Installation Media List Software Fixes (APARs) on Installation Media Manage Network Install Permissions Manage Network Install Resource Allocation	**nim_client** **niminit** **nim_client_inst** **nim_client_list** **nim_c_list_bundle** **nim_c_list_sw** **nim_c_list_fixes** **nim_perms** **nim_c_mac_res**
System Backup Manager Back Up the System Back Up This System to Tape/File Create a Generic Backup CD List Files in a System Image Restore Files in a System Image	**backsys** **sysbackup** **mksysb** **mkcdgeneric** **lsmksysb** **restmksysb**

Software License Management

Fast path to menu: **licenses**

Task	Fast Path
Manage Nodelocked Licenses Add Nodelocked License from a File Add Nodelocked License from the Keyboard Delete a Nodelocked License	**manage_nodelocked** **add_nodelocked_from_file** **add_nodelocked_from_keyboard** **delete_nodelocked**

Task	Fast Path
Manage License Servers and License Databases Show Server Characteristics Manage Concurrent Use and Use Once Licenses Manage Vendor Information in License Databases	**manage_servers** **show_server_characteristics** **manage_prod_licenses** **manage_vendors**
Show License Usage on Servers Show License Usage Summary Show Licenses Currently Being Used Show License Information by Server Show Licenses Held by a Specific User	**show_server_status** **show_total_license_usage** **show_current_license_usage** **show_installed_licenses** **show_user_license_held**
Show License Agreements Show Installed License Agreements Show License Agreements on Installation Media	**show_license_agree** **installed_license** **license_on_media**

Devices

Fast path to menu: **dev**

Task	Fast Path
Install/Configure Devices Added After IPL	**cfgmgr**
Printer/Plotter	**printer**
TTY	**tty**
PTY	**pty**
Console	**console**
Fixed Disk	**disk**
CD ROM Drive	**cdrom**
Read/Write Optical Drive	**rwopt**
Diskette Drive	**diskette**
Tape Drive	**tape**
Communication	**commodev**
Graphic Displays	**g_display**
Graphic Input Devices	**input**
Low Function Terminal (LFT)	**lft**
SCSI Initiator Device	**scsiid**
SCSI Adapter	**scsia**
Asynchronous I/O	**aio**
Multimedia	**mm**
List Devices	**lsattr**
Configure/Unconfigure Devices Unconfigure a Device Configure a Defined Device	**devcfg** **devcfg_ucfg** **devcfg_cfg**
Install Additional Device Software	**devinst**

Task	Fast Path
PCI Hot Plug Manager Unconfigure a Device Configure a Defined Device Install/Configure Devices Added After IPL	**devdrpci** **rmdev** **mkdev** **cfgmgr**
ISA Adapters	**devisa**

System Storage Management

Fast path to menu: **storage**

Task	Fast Path
Logical Volume Manager	**lvm**
Volume Groups List All Volume Groups Add a Volume Groups Set Characteristics of a Volume Group List Contents of a Volume Group Remove a Volume Group Activate a Volume Group Deactivate a Volume Group Import a Volume Group Export a Volume Group Mirror a Volume Group Unmirror a Volume Group Synchronize LVM Mirrors Back Up a Volume Group Remake a Volume Group List Files in a Volume Group Backup Restore Files in a Volume Group Backup	**vg** **lsvg2** **mkvg** **vgsc** **lsvg1** **reducevg2** **varyonvg** **varyoffvg** **importvg** **exportvg** **mirrorvg** **unmirrorvg** **syncvg** **vgbackup** **restvg** **lsbackvg** **restsavevg**
Logical Volumes List All Logical Volumes by Volume Group Add a Logical Volume Set Characteristics of a Logical Volume Show Characteristics of a Logical Volume Remove a Logical Volume Copy a Logical Volume	**lv** **lsvg** **mklv** **lvsc** **lslv** **rmlv** **cplv**
Physical Volumes Add a Disk Change Characteristics of a Physical Volume List Contents of a Physical Volume Move Contents of a Physical Volume	**pv** **makdsk** **chpv** **lspv** **migratepv**

Task	Fast Path
Paging Space	**pgsp**
Add Another Paging Space	**mkps**
Change/Show Characteristics of a Paging Space	**chps**
Remove a Paging Space	**rmps**
Activate a Paging Space	**swapon**
Deactivate a Paging Space	**swapoff**
File Systems	**fs**
List All File Systems	**lsfs**
List All Mounted File Systems	**mount**
Add/Change/Show/Delete File Systems	**manfs**
Mount a File System	**mountfs**
Mount a Group of File Systems	**mountg**
Unmount a File System	**umountfs**
Unmount a Group of File Systems	**umountg**
Verify a File System	**fsck**
Backup a File System	**backfilesys**
Restore a File System	**restfilesys**
List Contents of a Backup	**listtoc**
Files & Directories	**filemgr**
Backup a File or Directory	**backfile**
Restore a File or Directory	**restfile**
List Contents of a Backup	**listtoc**
Removable Disk Management	**rds**
List All Mounted File Systems on a Disk	**lsmntdsk**
Unmount File Systems on a Disk	**umntdsk**
Remove a Disk from the Operating System	**removedsk**
Remove a Disk	**rmvdsk1**
Open Door	**open_door**
System Backup Manager	**backsys**
Back Up the System	**sysbackup**
List Files in a System Image	**lsmksysb**
Restore Files in a System Image	**restmksysb**

Security & Users

Fast path to menu: **security**

Task	Fast Path
Users	**users**
Add a User	**mkuser**
Change a User's Password	**passwd**
Change/Show Characteristics of a User	**chuser**
Lock/Unlock a User's Account	**lockuser**
Reset User's Failed Login Count	**failed_logins**
Remove a User	**rmuser**
List All Users	**lsuser**

Task	**Fast Path**
Groups List All Groups Add a Group Change/Show Characteristics of a Group Remove a Group	**groups** **lsgroup** **mkgroup** **chgroup** **rmgroup**
Passwords Change a User's Password Change/Show Password Attributes for a User	**passwords** **passwd** **passwdattrs**
Login Controls Change/Show Login Attributes for a User Change/Show Login Attributes for a Port	**logins** **login_user** **login_port**
Roles Add a Role Change/Show Characteristics of a Role Remove a Role List All Roles	**roles** **mkrole** **chrole** **rmrole** **lsrole**

Communications Applications and Services

Fast path to menu: **commo**

Task	**Fast Path**
TCP/IP	**tcpip**
Minimum Configuration & Startup	**mktcpip**
Further Configuration Hostname Static Routes Network Interfaces Name Resolution Client Network Services Server Network Services Manage Print Server Select BSD style rc Configuration Authentication Configuration	**configtcp** **hostname** **route** **netinterface** **namerslv** **clientnet** **ruser** **server** **setbootup_option** **auth_config**
Use DHCP for TCPIP Configuration & Startup	**usedhcp**
IPV6 Configuration IPV6 Static Routes IPV6 Network Interfaces IPV6 Daemon/Process Configuration	**configtcp6** **route6** **inet6** **daemon6**
Quality of Service Configuration & Startup Start Using the QoS Subsystem Stop Using the QoS Subsystem	**configqos** **startqos** **stopqos**
NFS	**nfs_menus**
Configure TCP/IP (If Not Already Configured)	**tcpip**

Task	Fast Path
Network File System (NFS) Configure NFS on This System Add a Directory to Exports List Change/Show Attributes of an Exported Directory Remove a Directory from Exports List Add a File System for Mounting Change/Show Attributes of an NFS File System Remove Remove an NFS File System	**nfs** **nfsconfigure** **mknfsexp** **chnfsexp** **rmnfsexp** **mknfsmnt** **chnfsmnt** **rmnfsmnt**

Print Spooling

Fast path to menu: **spooler**

Task	Fast Path
Start a Print Job	**qprt**
Manage Print Jobs Cancel a Print Job Show the Status of Print Jobs Prioritize a Print Job Hold/Release a Print Job Move a Job Between Print Queues	**jobs** **qcan** **qchk** **qpri** **qhld** **qmove**
Manage Print Queues Show Status of Print Queues Stop a Print Queue Start a Print Queue Set the System's Default Print Queue	**pqmanage** **qstatus** **qstop** **qstart** **qdefault**
Add a Print Queue	**mkpq**
Add an Additional Printer to an Existing Print Queue	**mkqprt**
Change/Show Print Queue Characteristics	**chpq**
Remove a Print Queue	**rmpq**
Manage Print Server	**server**
Programming Tools	**pqtools**

Problem Determination

Fast path to menu: **problem**

Task	Fast Path
Error Log Generate Error Report Change/Show Characteristics of the Error Log Clean the Error Log	**error** **errpt** **errdemon** **errclear**

Task	Fast Path
System Dump	**dump**
Change the Primary Dump Device	**dumpchgp**
Change the Secondary Dump Device	**dumpchgs**
Change the Directory to which Dump is Copied on Boot	**dumpchgd**
Copy a System Dump from a Dump Device to a File	**dump_copy_file**
Copy a System Dump from a Dump Device to Diskette	**dump_copy_dskt**
Always Allow System Dump	**dump_allow**
System Dump Compression	**dump_comprs**
Check Dump Resources Utility	**dump_checkr**
Alog	**alog**
Show an Alog file	**alog_show**
Change/Show Characteristics of an Alog File	**alog_change**
Hardware Diagnostics	**diag**
Verify Software Installation and Requisites	**verify_install**

Performance & Resource Scheduling

Fast path to menu: **performance**

Task	Fast Path
Resource Status & Monitors	**monitors**
Analysis Tools	**analysis**
Resource Controls	**controls**
Remove a Process	**kill**
Set Initial Priority of a Process	**nice**
Change Initial Priority of a Process	**renice**
Set System Run Level	**telinit**
Schedule Jobs	**at**
Power Management	**pm**
Configure/Unconfigure Power Management	**pmConfig**
System State Transition from Enable State	**pmState**
Display Power Management	**pmDisplaySelect**
Battery	**pmBattery**
Workload Management	**wlm**
Work on alternate configurations	**wlmconfig**
Copy a configuration	**wlmconfig_copy**
Create a configuration	**wlmconfig_create**
Select a configuration	**wlmconfig_select**
Enter configuration description	**wlmconfig_enter**
Remove a configuration	**wlmconfig_delete**
Work on a set of Subclasses	**wlmsubclass**
Add a class	**wlmaddclass**

Task	Fast Path
Change/Show Characteristics of a class General characteristics of a class CPU resource management Memory resource management diskIO resource management	**wlmchclass** **wlmclass_gal** **wlmclass_cpu** **wlmclass_mem** **wlmclass_bio**
Remove a class	**wlmrmclass**
Class assignment rules Create a new Rule Change/Show Characteristics of a Rule	**wlmrs** **crewlmrs** **chgwlmrs**
Start/Stop/Update WLM Start Workload Management Update Workload Management Stop Workload Management	**wlmmanage** **wlmstart** **wlmupdate** **wlmoff**
Assign/Unassign processes to a class/subclass	**wlmassign**

System Environments

Fast path to menu: **system**

Task	Fast Path
Stop the System	**system**
Assign the Console	**chcons**
Change/Show Date and Time Change/Show Date & Time Change Time Zone Using System Defined Values Change Time Zone Using User Inputted Values	**chtz_date** **date** **chtz** **chtz_user**
Manage Language Environment Change/Show Primary Language Environment Add Additional Language Environments Remove Language Environments Change/Show Language Hierarchy Set User Languages Change/Show Applications for a Language Convert System Messages and Flat Files	**mlang** **chlang** **mle_add_lang** **mle_rm_lang_hdr** **mle_hier_cmd_hdr** **chlang_user** **mle_chapp_menu** **nu_iconv**
Change/Show Characteristics of Operating System	**chgsys**
Change/Show Number of Licensed Users	**chlicense**
Manage AIX Floating User Licenses for this Server	**netls_server**
Broadcast Message to all Users	**wall**
Manage System Logs	**logs**
Change/Show Characteristics of System Dump	**dump**

Task	Fast Path
Internet and Documentation Services Change/Show Default Browser Change Documentation and Search Server Change/Show Default Documentation Language Web-based System Manager	**web_configure** **change_default_browser** **change_doc_search_server** **chdoclang** **web_based_system_manager**
Change System User Interface	**dt_config**
Change/Show Default Documentation Language	**chdoclang**
Manage Remote Reboot Facility	**rrbtty**
Manage System Hang Detection	**shd**

Processes & Subsystems

Fast path to menu: **src**

Task	Fast Path
Processes Remove a Process Bind a Process to a Processor Unbind a Process	**process** **kill** **bindproc** **unbindproc**
Subsystems Query a Subsystem Start a Subsystem Stop a Subsystem Stop a Single Subsystem Stop All Subsystems Refresh a Subsystem Trace Subsystem Start Trace Stop Trace	**subsys** **qssys** **startssys** **stopssys** **stopassys** **stopallssys** **refresh** **tracessys** **tracessyson** **tracessysoff**
Subservers Query a Subserver Start a Subserver Stop a Subserver Trace Subserver Start Trace Stop Trace	**subserver** **qserver** **startserver** **stopserver** **traceserver** **startserver.trace** **stopserver.trace**

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.