TXSeries™ for Multiplatforms

Encina® Administration Guide Volume 1: Basic Administration

Version 5.0

SC09-4473-01

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 275.

Second Edition (December 2001)

This edition replaces SC09-4473-00.

Order publications through your IBM representative or through the IBM branch office serving your locality.

Contents

Figures

Tables

About this book

This book focuses on configuring and administering Encina® systems.

Unless otherwise noted, information in this document applies to all supported platforms (UNIX™ and Windows NT™). Documentation that is qualified by *UNIX* provides information for all UNIX platforms, including AIX™; documentation that is qualified by *Non-AIX UNIX* provides information for all non-AIX UNIX platforms (such as Solaris™). Documentation that is qualified by *NT* provides information for the Windows NT platform.

Who should read this book

This document is written for Encina system administrators responsible for configuration and administration of Encina systems. This document assumes that readers are familiar with system administration, particularly for the platform they are working on. This document also assumes that readers are familiar with the Open Software Foundation® (OSF) Distributed Computing Environment (DCE) and have access to the DCE documentation set.

Document organization

This document has the following organization:

Part 1: *Administering servers in a Monitor cell*

- "Chapter 1. Using the Encina interfaces" on page 3 describes the graphical and command-line interfaces used for system configuration and day-to-day administrative tasks. It includes a summary of syntax for command-line interfaces.
- "Chapter 2. Administering servers in a Monitor cell" on page 31 describes how to perform basic administrative tasks for servers in a Monitor cell. These tasks include stopping and restarting cell and node managers, and defining and starting different types of servers.

Part 2: *Server maintenance and recovery*

- "Chapter 3. Creating volumes" on page 69 describes data storage in Encina. It contains procedures for creating and mirroring volumes, configuring complex volumes, and expanding volumes.
- "Chapter 4. Maintaining volumes" on page 95 contains procedures for relocating and renaming volumes, and for reclaiming storage space occupied by volumes. It also describes how to restart a server in administrative mode.

- "Chapter 5. Managing server transactions" on page 109 describes distributed transaction processing, how to use Enconsole to view information about transactions in a server, and how to take action on problematic unresolved transactions.
- "Chapter 6. Performing backups" on page 121 describes how to implement a secure and reliable backup policy. It describes Encina's media archiving, how to make backups, and how to manage and reclaim storage from older backups.
- "Chapter 7. Recovering from failures" on page 133 describes how to handle abnormal server terminations and media failures. It contains procedures for restoring both log and data volumes from backup files.

Part 3: *Server security and performance*

- "Chapter 8. Understanding and controlling access to Encina resources" on page 153 describes DCE access control lists (ACLs) and the interfaces used to create and modify them. It lists the permissions required for administrative operations and shows example commands for creating and modifying ACLs.
- "Chapter 9. Modifying server behavior" on page 183 describes server parameters that affect performance. It also describes how to modify the number of processing agents in a server, how to create server sets, and how to define schedules for servers.
- "Chapter 10. Using the Encina Trace Facility" on page 211 describes the format of Encina trace output, how to control the amount of trace output, how to enable and disable tracing, and how to specify the destination of trace output. This chapter also describes commands for dumping state information.

The Glossary defines terms associated with DCE and Encina.

Related information

For further information on the topics discussed in this manual, see the following documents:

- *Concepts and Planning*
- *Planning and Installation Guide*
- *Encina Administration Guide Volume 2: Server Administration*
- *Encina Administration Guide Volume 3: Advanced Administration*

Conventions used in this book

TXSeries documentation uses the following typographical and keying conventions.

Table 1. Conventions used in this book

Convention	Meaning
Bold	Indicates values you must use literally, such as commands, functions, and resource definition attributes and their values. When referring to graphical user interfaces (GUIs), bold also indicates menus, menu items, labels, buttons, icons, and folders.
`Monospace`	Indicates text you must enter at a command prompt. Monospace also indicates screen text and code examples.
Italics	Indicates variable values you must provide (for example, you supply the name of a file for *file_name*). Italics also indicates emphasis and the titles of books.
< >	Enclose the names of keys on the keyboard.
<**Ctrl-***x*>	Where *x* is the name of a key, indicates a control-character sequence. For example, <**Ctrl-c**> means hold down the **Ctrl** key while you press the **c** key.
<**Return**>	Refers to the key labeled with the word Return, the word Enter, or the left arrow.
`%`	Represents the UNIX command-shell prompt for a command that does not require **root** privileges.
`#`	Represents the UNIX command-shell prompt for a command that requires **root** privileges.
`C:\>`	Represents the Windows® command prompt.
>	When used to describe a menu, shows a series of menu selections. For example, "Select **File > New**" means "From the **File** menu, select the **New** command."
Entering commands	When instructed to "enter" or "issue" a command, type the command and then press <**Return**>. For example, the instruction "Enter the **ls** command" means type **ls** at a command prompt and then press <**Return**>.
[ ]	Enclose optional items in syntax descriptions.
{ }	Enclose lists from which you must choose an item in syntax descriptions.
|	Separates items in a list of choices enclosed in **{ }** (braces) in syntax descriptions.
...	Ellipses in syntax descriptions indicate that you can repeat the preceding item one or more times. Ellipses in examples indicate that information was omitted from the example for the sake of brevity.
IN	In function descriptions, indicates parameters whose values are used to pass data to the function. These parameters are not used to return modified data to the calling routine. (Do *not* include the IN declaration in your code.)
OUT	In function descriptions, indicates parameters whose values are used to return modified data to the calling routine. These parameters are not used to pass data to the function. (Do *not* include the OUT declaration in your code.)

Table 1. Conventions used in this book (continued)

Convention	Meaning
INOUT	In function descriptions, indicates parameters whose values are passed to the function, modified by the function, and returned to the calling routine. These parameters serve as both IN and OUT parameters. (Do *not* include the INOUT declaration in your code.)
$CICS	Indicates the full path name where the CICS product is installed; for example, **C:\opt\cics** on Windows or **/opt/cics** on Solaris. If the environment variable named CICS is set to the product path name, you can use the examples exactly as shown; otherwise, you must replace all instances of $CICS with the CICS product path name.
CICS on Open Systems	Refers collectively to the CICS product for all supported UNIX platforms.
TXSeries CICS	Refers collectively to the CICS for AIX, CICS for Solaris, and CICS for Windows products.
CICS	Refers generically to the CICS on Open Systems and CICS for Windows products. References to a specific version of a CICS on Open Systems product are used to highlight differences between CICS on Open Systems products. Other CICS products in the CICS Family are distinguished by their operating system (for example, CICS for OS/2 or IBM mainframe-based CICS for the ESA, MVS, and VSE platforms).

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other WebSphere Application Server Enterprise Edition documentation, send your comments by e-mail to wasdoc@us.ibm.com. Be sure to include the name of the book, the document number of the book, the version of WebSphere Application Server Enterprise Edition, and, if applicable, the specific location of the information you are commenting on (for example, a page number or table number).

Part 1. Administering servers in a Monitor cell

Chapter 1. Using the Encina interfaces

This chapter introduces the tools available for system configuration and day-to-day administrative tasks. It describes how to use the Enconsole graphical user interface (GUI) and briefly describes the Encina command-line and Distributed Computing Environment (DCE) interfaces, as well as the Tivoli Global Enterprise Manager (GEM).

Introduction to the Encina interfaces

Enconsole is the primary administrative tool for Encina. Enconsole automates many of the initial DCE and Toolkit administrative tasks needed to configure and start servers. For example, creating and mirroring volumes can be automatically done with Enconsole. Other administrative tasks (such as creating backups) must be done outside of Enconsole by using the command-line interfaces. Once an Encina server is started, you administer it using server-specific interfaces. For example, the **rqsadmin** interface allows you to manage all Recoverable Queueing Service (RQS) objects, including queues, queue sets, and RQS servers. Table 2 lists the graphical and command-line interfaces used for various administrative tasks.

Table 2. Using Enconsole and the command-line interfaces

Administrative Task	**Interfaces Used**
Configuring and starting a Monitor cell	Enconsole
Defining, starting, and stopping servers	Enconsole
Backing up volumes	**tkadmin**
Moving, restoring, and renaming volumes	Enconsole and **tkadmin**
Creating and modifying ACLs for server objects	**dcecp** (or **acl_edit**)
Managing queues, queue sets, and element types in an RQS server	**rqsadmin**
Managing files and indices in an SFS server	**sfsadmin**
Administering a PPC Gateway server	**ppcadmin**
Administering a DE-Light Gateway server	**drpcadmin**
Writing administrative scripts	**enccp**

The Enconsole and command-line interfaces are described in the following sections.

Using Enconsole

This section provides an overview of Enconsole administrative capabilities and describes how Enconsole provides access to the Encina Monitor, the Encina Toolkit, and DCE. This section also introduces the Enconsole interface. The interface includes menus, which you use to direct Enconsole to perform operations; forms, which you use to specify the information necessary for Enconsole to perform operations; and display screens, which provide status information about servers and the transactions they are processing.

Distributed administration with Enconsole

Enconsole provides a *window* into the distributed Encina system. Figure 1 on page 5 provides an example of how the Enconsole interface displays information about your Monitor cell. The Encina Servers display screen shows all servers; the Nodes display screen shows all nodes; and the Serious Messages display screen shows all system messages. You can view display screens by selecting commands from the Enconsole menu. Using Enconsole's comprehensive view of the Encina system, you can monitor the system and troubleshoot problems from any network location.

You can monitor a wide variety of Encina client/server applications with Enconsole, including Recoverable Queueing Service (RQS), Structured File Server (SFS), and Peer-to-Peer Communications (PPC) applications. In addition, Enconsole lets you monitor DCE application servers, Toolkit servers, and *generic* servers. Generic servers are servers that do not rely on Encina or DCE capabilities and can be built with shell programs or the C programming language.

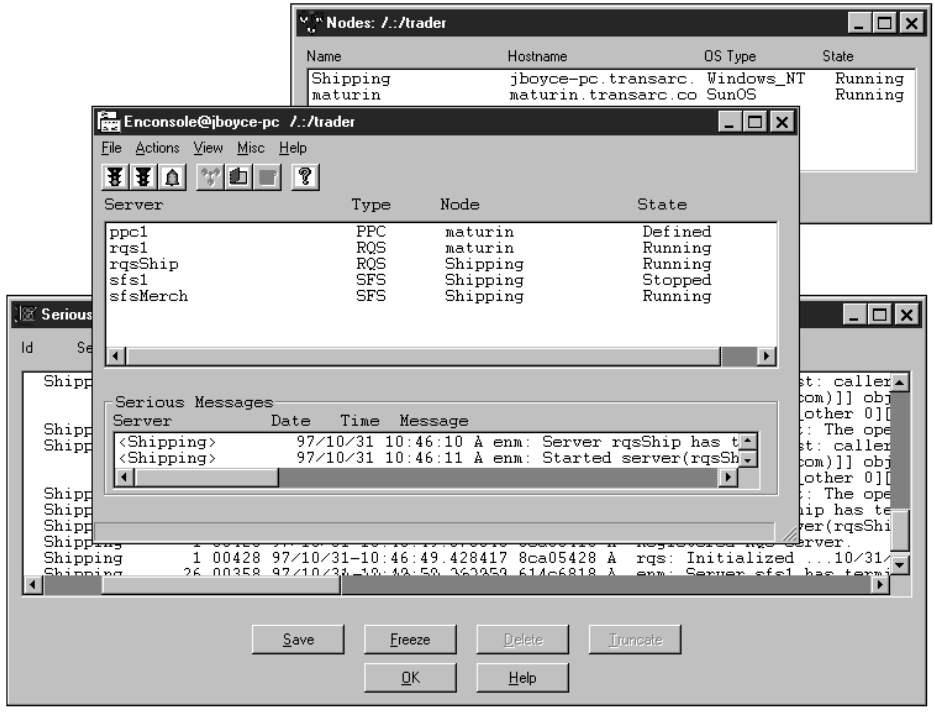

Figure 1. Enconsole: A window into the Encina system

Because Enconsole is based on the open, published Encina application programming interfaces (APIs), you can use it to administer a heterogeneous environment that includes a range of platforms, standards, and protocols. Note that Figure 1 shows a distributed system that includes machines running the SunOS and Windows NT operating systems.

Tools for defining servers

In addition to viewing information, you can use Enconsole to perform tasks such as defining, modifying, and deleting server configurations. Enconsole displays the forms necessary for configuring a server and its associated components, such as log volumes and data volumes.

Enconsole uses reasonable default values to ensure consistency throughout the network and to minimize the number of configuration choices that you must make. Enconsole supplies default operating system parameters such as the user ID, group ID, and working directory for a server. The Process Options form in Figure 2 on page 6 shows the default values for an RQS server's operating system attributes. Enconsole also supplies default values for the *advanced options* for each server. These advanced options are hidden unless

you request that they be displayed. Figure 2 shows that the RQS advanced options include the thread pool size, collating language, and key table options.

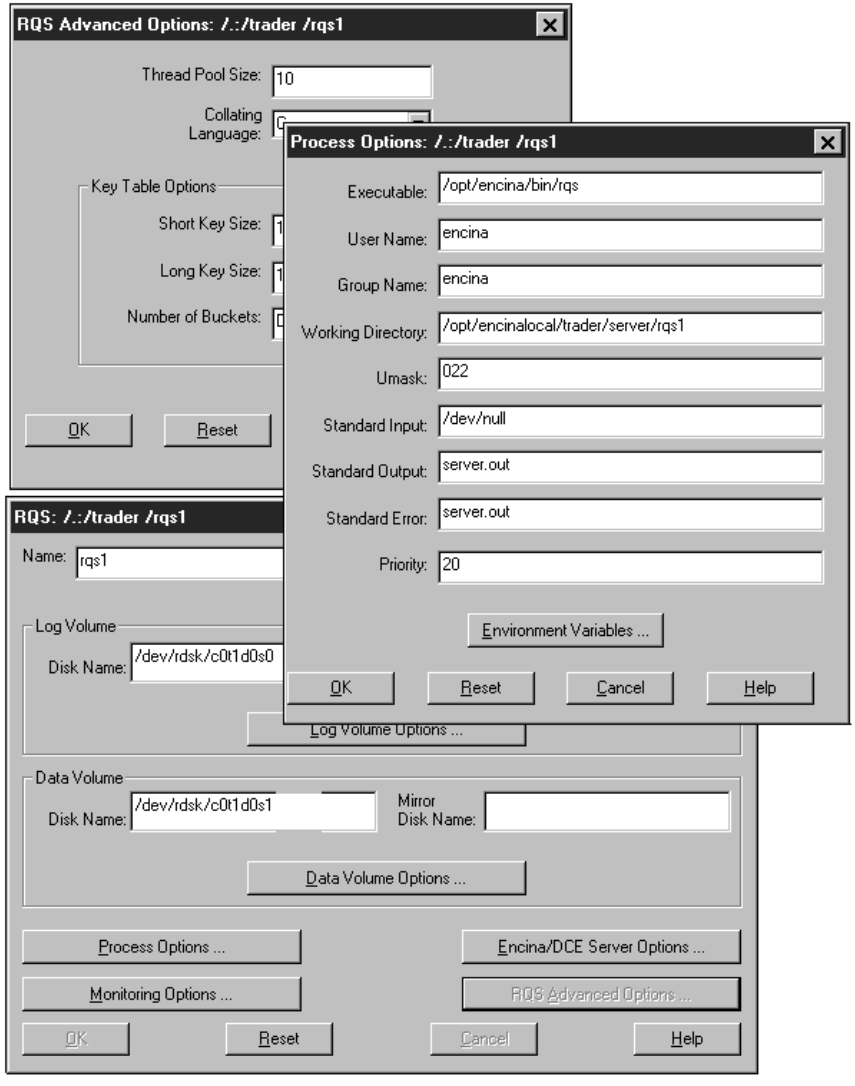

Figure 2. RQS, Process Options, and RQS Advanced Options forms

Enconsole provides access to the DCE settings for each application server. The principal ID and keytab file settings allow a server to establish and maintain its identity, providing protection from misuse and tampering. Figure 3 on page 7

page 7 illustrates the Encina/DCE Server Options form displaying the principal, keytab file, and Cell Directory Service (CDS) pathname fields.

Enconsole also provides access to the Encina settings for each server. You use Encina trace options to control the type of tracing and destination of trace output. By default, fatal, warning, and audit messages are displayed in the Enconsole Serious Messages display screen. This communication helps you reduce downtime by making it easier to recognize and respond to errors. Figure 3 shows that the Encina/DCE Server Options form displays the protection level, authorization, transaction inactivity timeout, and trace fields.

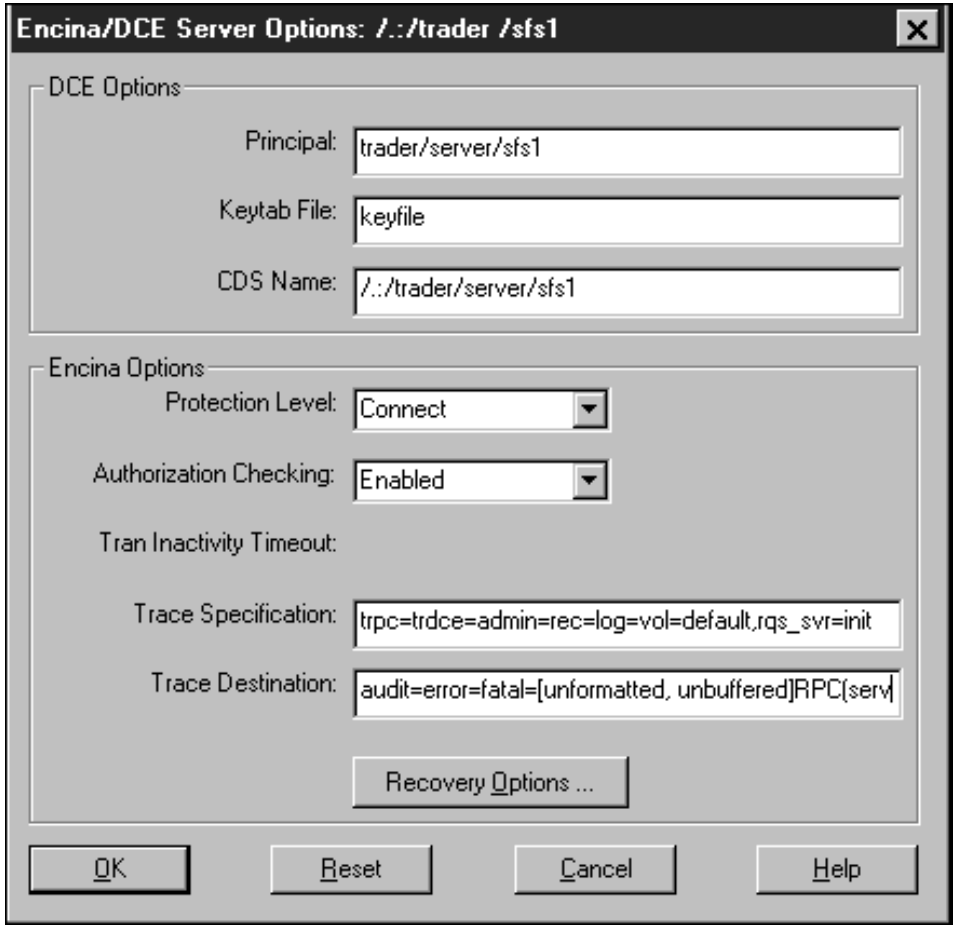

Figure 3. Encina/DCE Server Options form

Tools for starting and stopping servers

You can use Enconsole to start and stop servers individually from any node in the Monitor cell. You can also monitor the startup and shutdown messages for all servers. Figure 4 shows that you can use Enconsole commands to start cells, nodes, servers, and server sets. The Command Status display screen in Figure 4 shows the messages that Enconsole can display when you start an RQS server for the first time.

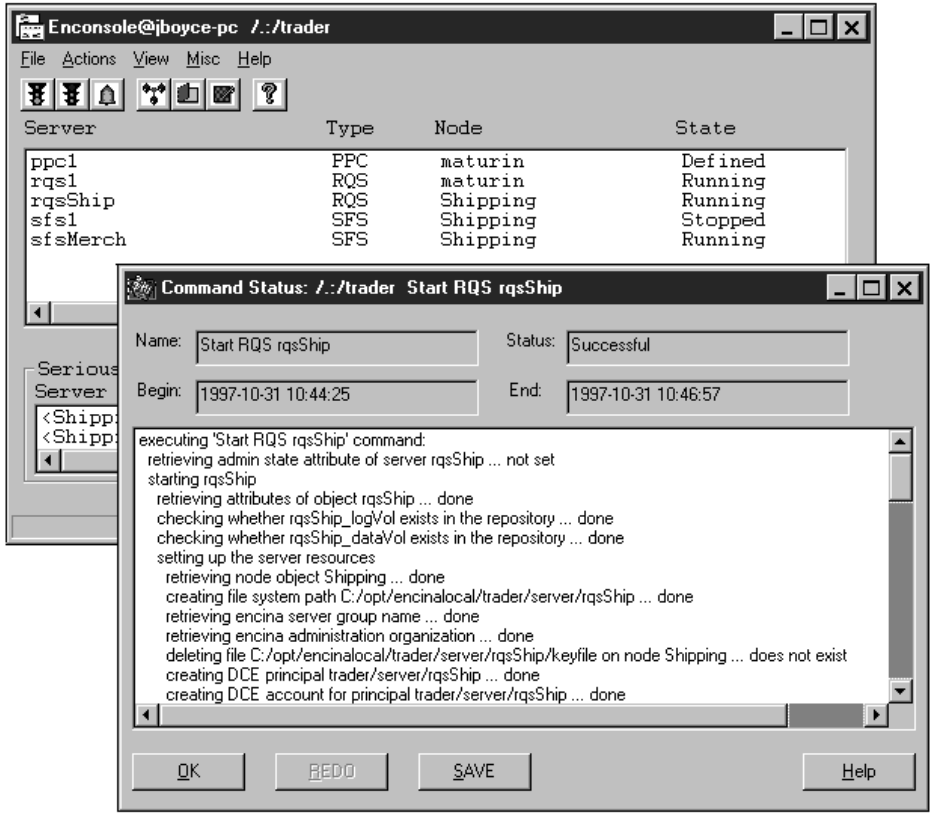

Figure 4. Startup menus and command status

Enconsole also provides ways to automate certain system administration tasks. You can use Enconsole to create a *server set* that starts or stops a collection of servers, or you can create *schedules*—plans that specify times at which to start or stop servers. Figure 5 on page 9 shows a schedule that automatically starts a server set named **merchSet** every morning and stops it every evening.

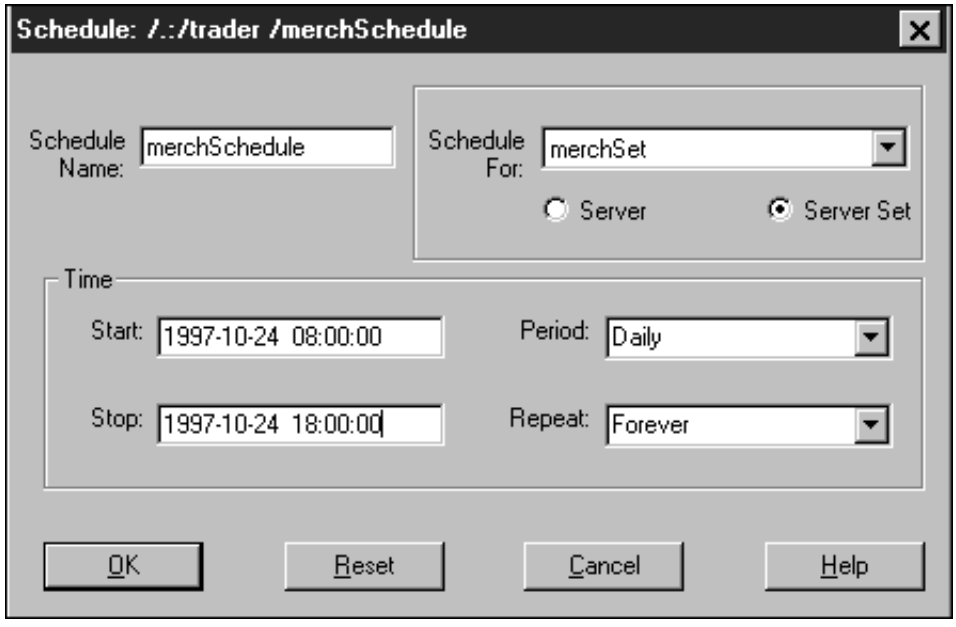

Figure 5. Daily schedule

Tools for monitoring and maintaining transactions

Enconsole displays information about the transactions processed by a server from any node in the Monitor cell. For any transaction, you can use Enconsole to review the transaction identifier of the transaction, number of locks held by the transaction, number of locks the transaction is awaiting, and state of the transaction. You can also review information about the parent transaction and Remote Procedure Calls (RPCs) associated with the transaction.

Enconsole also provides ways to resolve hung transactions. You can abort unresolved transactions, and you can release all locks held by a prepared transaction and force the transaction to abort, commit, or finish. Figure 6 on page 10 shows the Server Transactions display screen and Figure 7 on page 11 shows the Transaction Detail display screen.

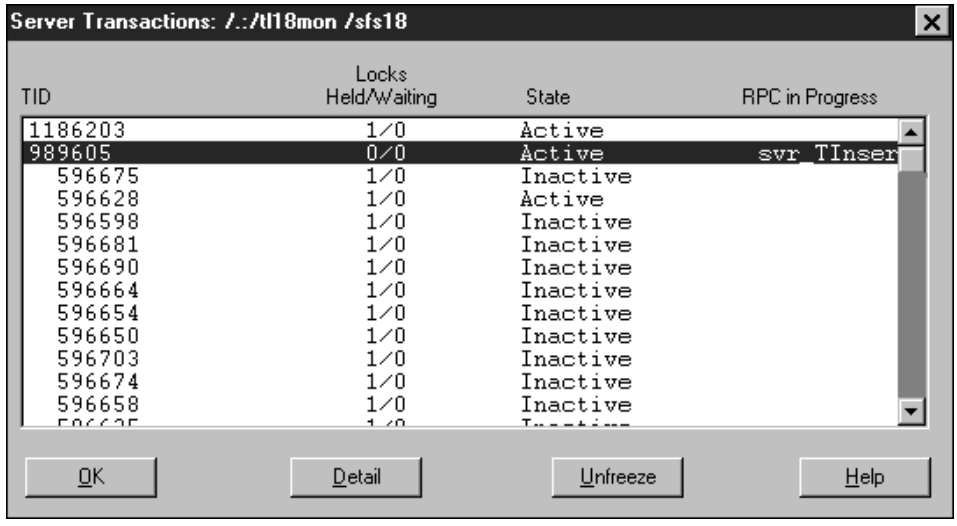

Figure 6. Server Transactions display screen

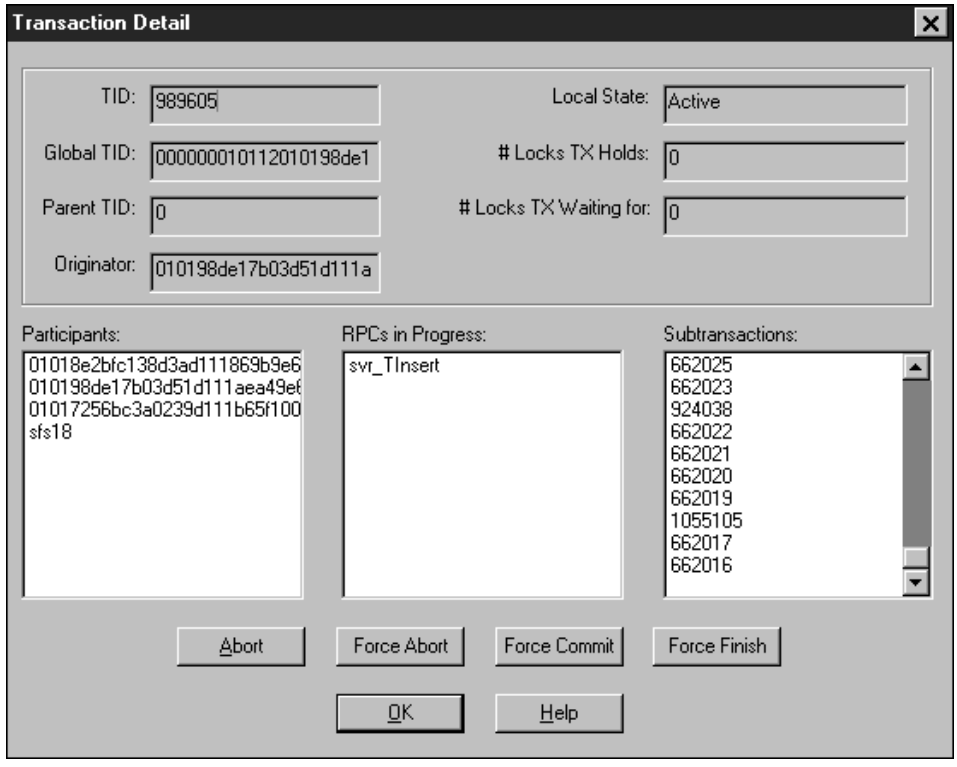

Figure 7. Transaction Detail display screen

The Encina Servers display screen

The Encina Servers display screen shows the list of servers that are defined in the current Monitor cell. This display screen also shows the serious messages that have been forwarded by the cell manager during the current Enconsole session. You can use this screen to monitor, but not manipulate, servers. Figure 8 on page 12 provides an example of the Encina Servers display screen.

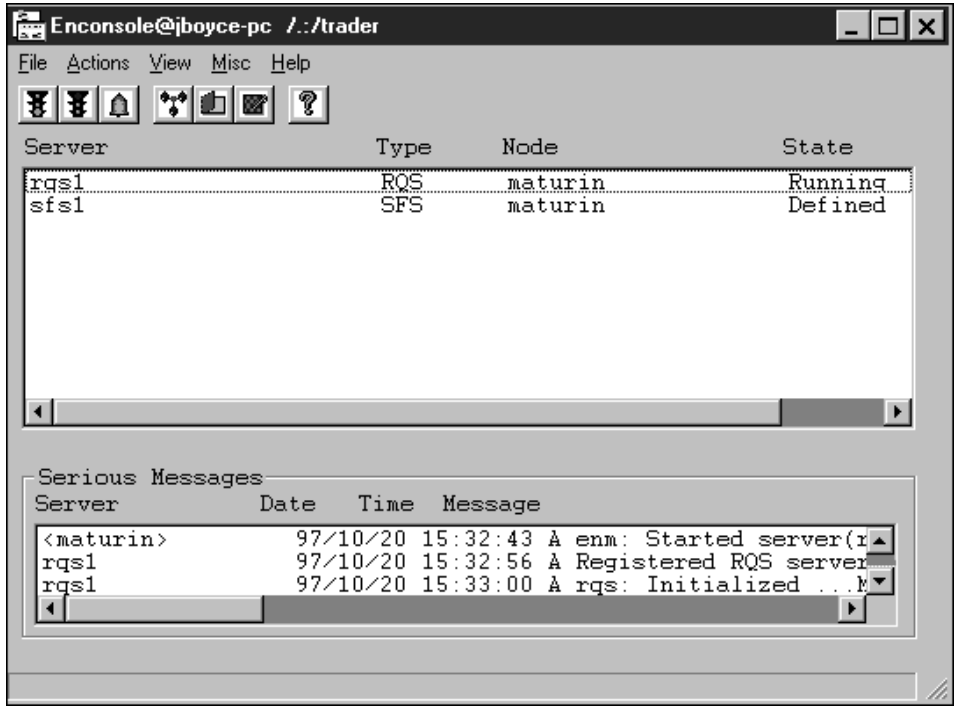

Figure 8. Encina Servers display screen

Menus

The **Encina Console** menu displays the following list of menus from which you can choose Enconsole commands:

- **File** — Each command manipulates the Enconsole session.
- **Actions** — Each command allows you to choose an action.
- **View** — Each command displays a specified display screen.
- **Misc** — Each command performs a DCE task.
- **Help** — Each command provides access to online help.

You access a menu by pressing and holding the left mouse button. To select a menu item, drag the mouse until the item is highlighted and then release the left mouse button.

An arrow to the right of a menu item provides access to an additional, drop-down menu. Figure 9 on page 13 shows the **Actions** menu with the **Define** command selected. The **Define** command in turn provides access to a

drop-down menu in which the **Server** command is selected; the **Server** command provides a menu of servers that can be defined.

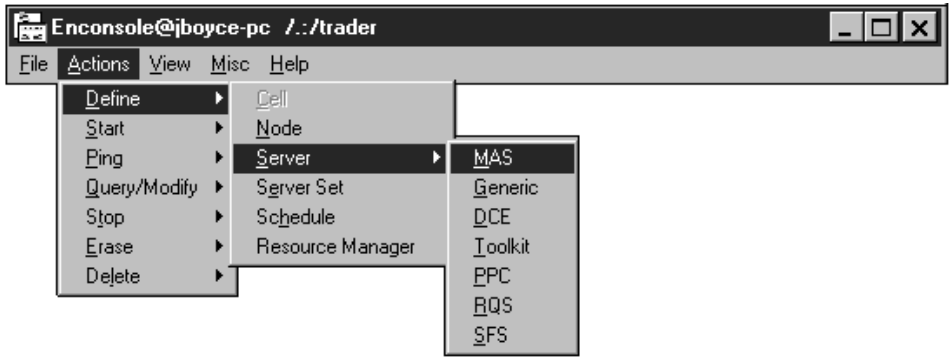

Figure 9. Enconsole menus

Forms

You use forms to provide information to Enconsole. As shown in Figure 10 on page 14, a form can contain option lists, fields, and buttons. When you use forms, you supply information in one of the following ways:

- By selecting a value from an option list. If a field has an option list, Enconsole displays an arrow to the right of the field. To use an option list, select the arrow, and then select a value from the drop-down list that is displayed.
- By typing a value into a field. You can always type values directly into any field. Select the field and then type the information.

Option lists, fields, and buttons

When you use a form, such as the SFS form shown in Figure 10 on page 14, you can specify information in fields by either typing in a value or selecting a value from an option list. In the figure, the **Node** field provides an option list from which you can select the node on which to run the SFS server. The **Mirror Disk Name** field and the other fields that accept disk names do not provide option lists, so you must type values directly into those fields.
Buttons, such as the **Help** button, initiate actions. The common buttons that are available on many Enconsole forms allow you to perform the following actions:

- The **OK** button saves all changes and closes the form.
- The **RESET** button discards all changes made since the form was opened.
- The **CANCEL** button discards any changes and closes the form.
- The **HELP** button displays help for the form.

As shown in Figure 10, server forms are used to define servers. They also provide access to other related forms. The following section provides more information about related forms.

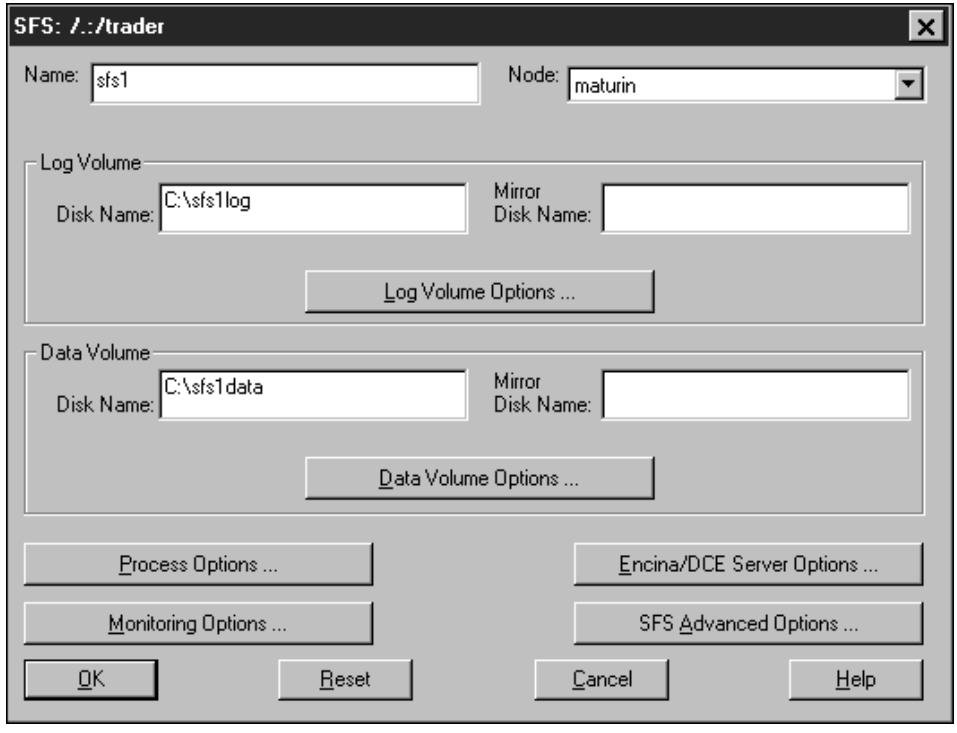

Figure 10. Option lists, fields, and buttons

Related forms

You can access additional related forms for each server by choosing option buttons from the server form. For example, the SFS server form in Figure 10 includes the **Encina/DCE Server Options** button. Notice that the text on the **Encina/DCE Server Options** button is followed by an **...** (ellipsis) to indicate that the button provides access to another form. When you choose this button, Enconsole displays the Encina/DCE Server Options form. Figure 11 on page 15 provides an example. You can use this form to manipulate the DCE- and Encina-related attributes for a server and to access the Recovery Options form via the **Recovery Options** button.

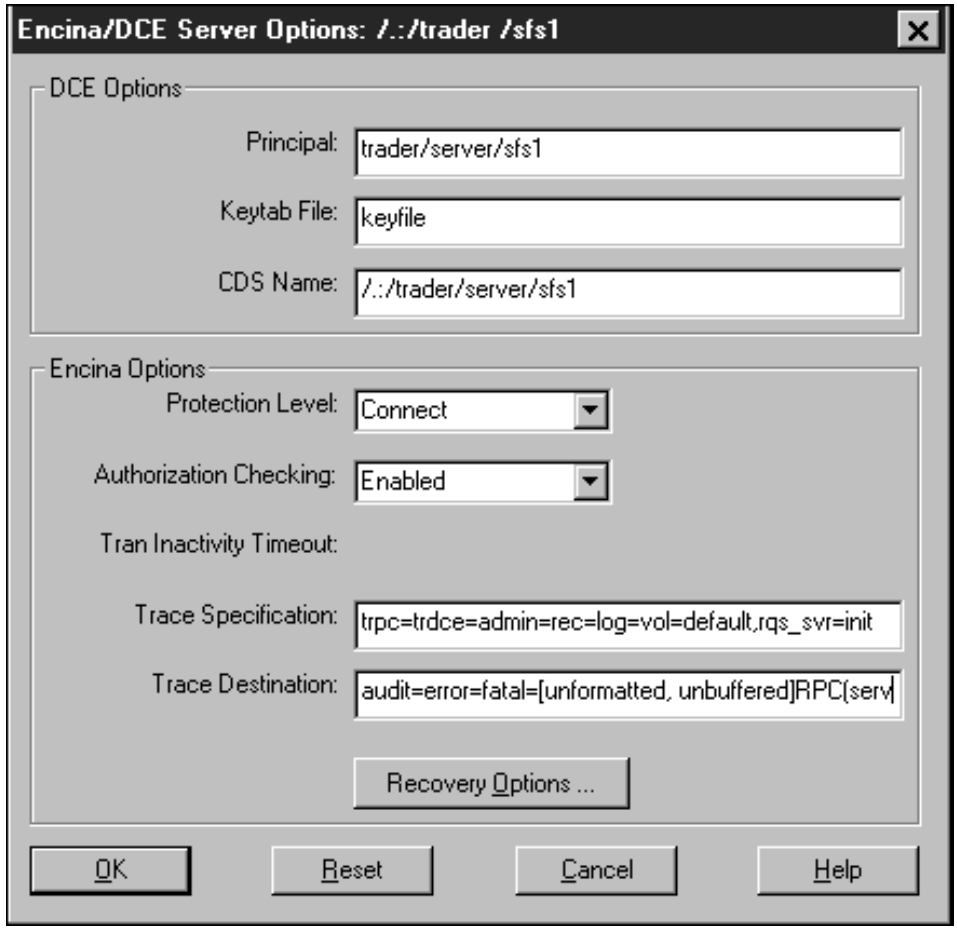

Figure 11. Related forms available from the Server Form

Selection Boxes

A selection box shows the objects (servers, nodes, and so forth) that can be manipulated by a selected command. For example, you can apply the **Stop** command to servers that have the states `Starting`, `Configuring`, and `Running`. When you select the **Stop** command, a selection box displays only the names of the servers that can be stopped. Figure 12 on page 16 shows a selection box with a list of servers that can be stopped. When you select an object and choose the **OK** button, the system closes the box and executes the command on the selected object.

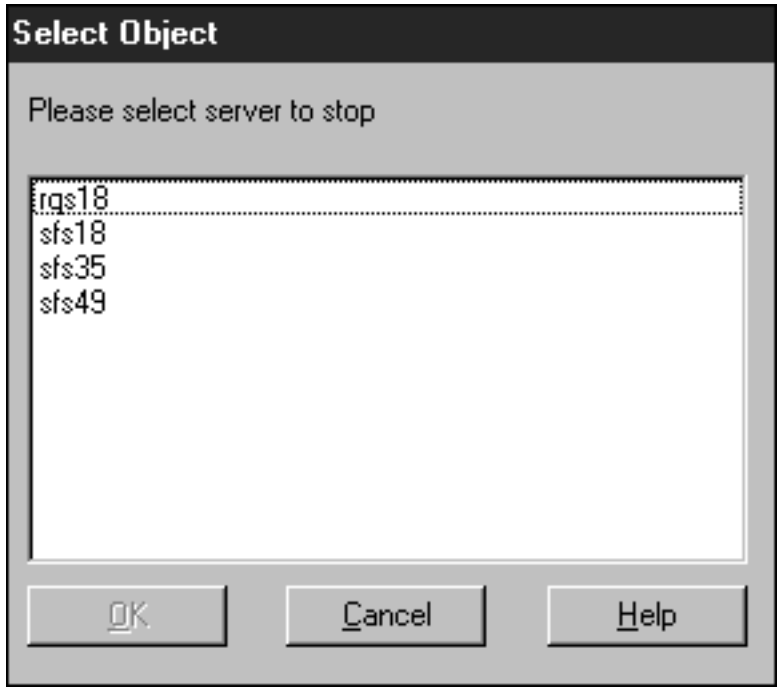

Figure 12. Selection box

Display screens

Display screens provide dynamic read-only information about servers, nodes, commands, and transactions. Figure 13 on page 17 shows the Server Transactions display screen. You can use the scroll bar to the right of the display field to view information that does not fit inside the display screen. You can select the up or down scroll arrow to move up or down one line. You can drag the scroll box in the scroll bar; the section of the list that moves into view depends on where you position the scroll box. For example, if you position the scroll box at the top of the scroll bar, Enconsole displays the text at the top of the list.

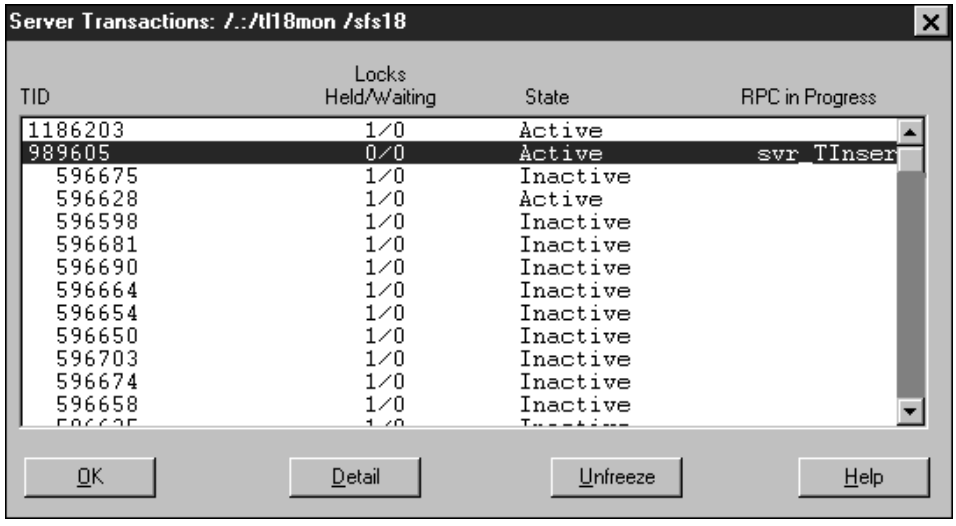

Figure 13. Server Transactions display screen

By default, Enconsole display screens are updated continually. As shown in Figure 13, on some display screens you can stop the updating by choosing the **Freeze** button. When the display screen has been frozen, the **Freeze** button toggles to **Unfreeze**. Choosing **Unfreeze** causes Enconsole to resume updating the screen.

When Enconsole has been idle for a specified amount of time, it automatically goes into dormant mode. The menu titles change to reflect the dormant mode, and Enconsole stops showing updated information on the display screens. As soon as you click a button or access a menu, Enconsole returns to its active state. You can specify when Enconsole enters the dormant mode by setting the ENCINA_ENCONSOLE_IDLE_INTERVAL environment variable, as described in the *Encina Administration Guide Volume 3: Advanced Administration*.

Online help

Online help is available for Enconsole display screens, forms, and fields. The following types of information are provided:

- Information on how to use online help
- General information about Enconsole
- Descriptions of display screens and forms
- Descriptions of fields
- Procedures that explain how to perform tasks

To access general Enconsole online help, choose the **Contents** command from the Enconsole **Help** menu. The Enconsole online help opens in a new window

and the "Welcome to Enconsole online help topic" is displayed. This topic provides general information on using the online help.

To access information about individual Enconsole display screens, forms, and fields, do one of the following:

- Choose the **Help** button on that display screen or form
- Press **<F1>** on that display screen or form

The Enconsole online help displays help text for that screen, including links to descriptions of each field that appears on the screen. Figure 14 illustrates the online help that is displayed when you click **Help** or press **<F1>** on the SFS Server form.

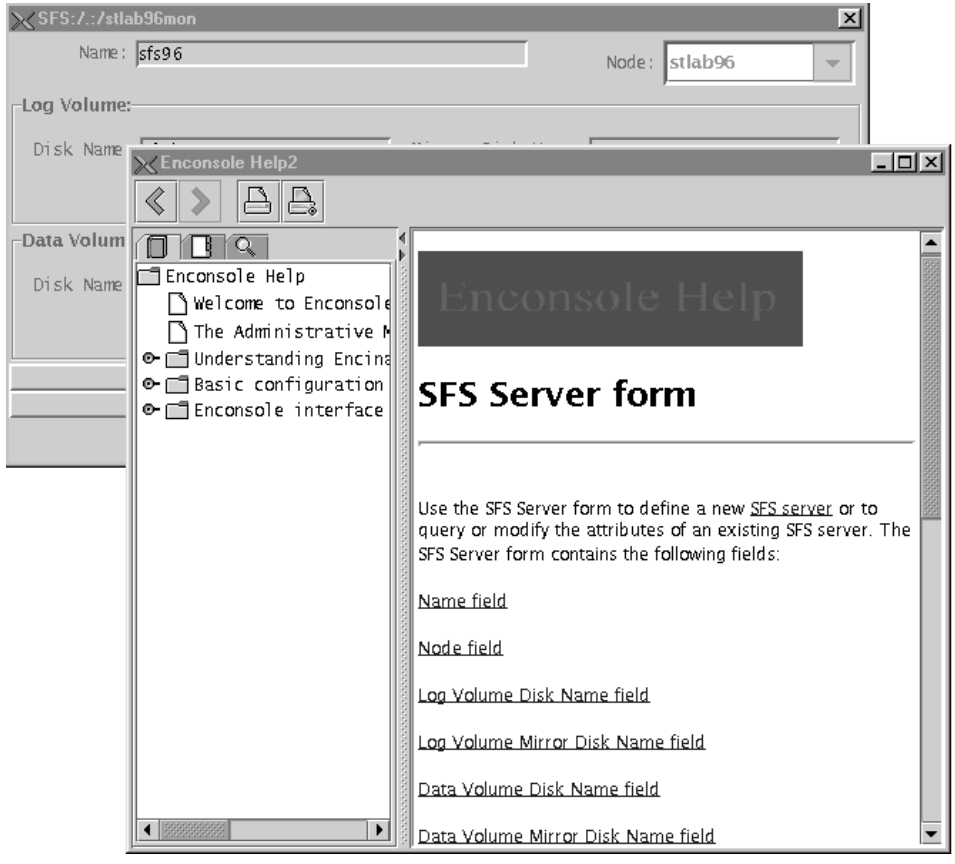

Figure 14. Help for the SFS Server form

To search the entire help text for a specific word or phrase, click the magnifying glass icon at the top of the left frame of the help application. In

the **Find** field, type the word or phrase to want to find, and then press **<Enter>**. The left frame displays a list of all topics that contain the specified word or phrase.

Enconsole keyboard commands

You can use the mouse or keyboard to choose Enconsole commands. Table 3 shows the keyboard operations available for various terminal types. You can use either uppercase or lowercase letters with the keyboard commands in Table 3.

Table 3. Enconsole keyboard commands

Action	X or xterm	VT Series	Windows NT
OK	F1	PF1, Do	
Cancel	F2	PF2	Esc
Reset	F3	PF3	
Screen Help	F4	PF4, Help	F1
Field Help	Esc-h	Esc-h	
Select Command	Return	Return	Return
Move Cursor Left	Left Arrow, Ctrl-l	Left Arrow, Ctrl-l	Left Arrow
Move Cursor Right	Right Arrow, Ctrl-r	Right Arrow, Ctrl-r	Right Arrow
Move Cursor Up	Up Arrow, Ctrl-u	Up Arrow, Ctrl-u	Up Arrow
Move Cursor Down	Down Arrow, Ctrl-d	Down Arrow, Ctrl-d	Down Arrow
Scroll Field Left	Esc-l	Esc-l	Left Arrow
Scroll Field Right	Esc-r	Esc-r	Right Arrow
Page Up	Page Up, Esc-u	Esc-u	Page Up
Page Down	Page Down, Esc-d	Esc-d	Page Down
First Field	Home, F5, Esc-<	F17, Esc-<	
Previous Field	F6, Esc-Tab	F18, Esc-Tab	Shift-Tab
Next Field	F7, Tab	F19, Tab	Tab
Last Field	End, F8, Esc->	F20, Esc->	
Delete Character Left	Backspace	Backspace	Backspace
Delete Character Right	Delete	Delete	Delete
Delete Line	Esc-k	Esc-k	
Move to Menu	Esc- -	Esc- -	Alt
Move from Menu	Esc-m	Esc-m	Alt

Table 3. Enconsole keyboard commands (continued)

Action	X or xterm	VT Series	Windows NT
Cycle through Screens	Esc-c	Esc-c	
Zoom Window	Esc-z	Esc-z	
Refresh Screen	Esc- .	Esc- .	

Using the Encina command-line interfaces

This section briefly describes the Encina command-line interfaces. These interfaces are used to perform maintenance tasks such as removing, restoring, and renaming volumes. They are also used to perform server-specific administrative tasks such as creating queues for an RQS server.

Using the tkadmin command-line interface

The **tkadmin** command-line interface can be used for a wide range of volume management, transaction administration, and application tracing tasks. It can be used for basic tasks such as creating and mirroring volumes, performing daily backups, and querying server attributes. It can also be used for less-frequent tasks, such as renaming volumes and restoring data.

Many initial tasks required for defining servers (such as creating, allocating, and mirroring volumes) are performed when you complete the fields on server definition forms in Enconsole. Other tasks must be done exclusively with the **tkadmin** interface (or with a combination of Enconsole and the **tkadmin** interface).

Table 4 on page 21 shows administrative tasks, their corresponding **tkadmin** commands, and whether or not the task is automated in Enconsole. The **tkadmin** commands are always available, whether or not you choose to use Enconsole. The two interfaces are compatible; **tkadmin** commands (as well as other Encina command suites) can be used to administer a server started with Enconsole.

Note: On non-AIX UNIX systems, volumes can be stored on disks. On AIX systems, volumes can be stored in operating system logical volumes or on disks. On Windows NT systems, volumes can be stored on physical disks, on logical disk drives, or in operating system files.

Table 4. Using Enconsole and the Encina tkadmin interface

Administrative Task	Encina tkadmin Commands	Is this task automated in Enconsole?	Comments
Backing up volumes	**tkadmin backup lvol**	no	
	tkadmin query backup	no	
	tkadmin retain backups	no	
	tkadmin truncate backup	no	
Creating physical and logical volumes; adding volumes	**tkadmin init disk**	yes	
	tkadmin create pvol	yes	
	tkadmin create lvol	yes	
	tkadmin enable lvol	yes	
Relocating physical volumes	**tkadmin move pvol**	no	Server must be in administrative mode.
Renaming physical and logical volumes	**tkadmin rename pvol**	no	Server must be in administrative mode. If AIX operating system logical volumes are in use, rename them by using the AIX Logical Volume Manager, and then use the **tkadmin unmap lvol** and **tkadmin remap lvol** commands.
	tkadmin rename lvol	no	
	tkadmin unmap lvol (AIX logical volumes only)	no	
	tkadmin remap lvol (AIX logical volumes only)	no	

Table 4. Using Enconsole and the Encina tkadmin interface (continued)

Administrative Task	Encina tkadmin Commands	Is this task automated in Enconsole?	Comments
Expanding volumes	**tkadmin expand pvol**	yes	If AIX operating system logical volumes are in use, first expand them by using the AIX Logical Volume Manager, and then use Enconsole or the **tkadmin expand lvol**command to expand the Encina logical volume.
	tkadmin expand lvol	yes	
Reclaiming space	**tkadmin remove mirror**	no	If AIX operating system logical volumes are in use, unmap them by using the **tkadmin unmap lvol** command, then delete them by using the AIX Logical Volume Manager.
	tkadmin delete lvol	no	
	tkadmin delete pvol	no	
	tkadmin delete disk	no	
	tkadmin unmap lvol (AIX logical volumes only)	no	
Creating and synchronizing mirrors	**tkadmin create pvol**	yes	On Windows NT, either Windows NT operating system or Encina can be used for mirroring. If AIX operating system logical volumes are in use, create mirrors by using the AIX Logical Volume Manager.
	tkadmin add mirror	yes	
	tkadmin sync mirrors	yes	

Table 4. Using Enconsole and the Encina tkadmin interface (continued)

Administrative Task	Encina tkadmin Commands	Is this task automated in Enconsole?	Comments
Repairing mirrors	**tkadmin sync mirrors**	no	If AIX operating system logical volumes are in use, repair mirrors by using the AIX Logical Volume Manager.
Restoring data volumes	**tkadmin remove mirror**	no	Server must be in administrative mode.
	tkadmin restore lvols	no	
	tkadmin query restore	no	
	tkadmin enable archfile	no	
	tkadmin recover lvols	no	
Restoring log volumes	**tkadmin restore logvol**	no	Server must be in administrative mode.
	tkadmin enable archfile	no	
	tkadmin enable logfile	no	
Tracing	All tkadmin trace-related commands	no	Audit, Error, and Warning messages are automatically sent to Serious Messages display screen in Enconsole. Other tracing operations can be done using either Enconsole or **tkadmin** commands.

Note: To perform certain administrative tasks such as restoring volumes, the server must be in administrative mode—that is, the volumes that are being worked on must not be available for I/O. You must first start a server in administrative mode by using Enconsole and then use

tkadmin commands to administer the server. See the procedure in "Using administrative mode" on page 95 for details.

Reference pages for the entire **tkadmin** command suite are provided online in HTML.

Using server-specific command-line interfaces

The following server-specific command-line interfaces allow you to manipulate server objects after a server is started:

- **rqsadmin** — Use the **rqsadmin** interface for RQS servers to create queues, organize them into queue sets, and specify their priorities and service levels. This interface is also used for defining element types for queues, specifying how and when to perform maintenance tasks in the queue storage area, and adjusting other storage factors that affect performance in an RQS server.
- **sfsadmin** — Use the **sfsadmin** interface for SFS servers to create and manage SFS files and indices.
- **ppcadmin** — Use the **ppcadmin** interface for administering PPC Gateway servers.
- **drpcadmin** — Use the **drpcadmin** interface for administering DE-Light Gateway servers.

All server-specific command-line interfaces are described in the *Encina Administration Guide Volume 2: Server Administration*. Reference pages for each interface are provided online in HTML.

Command syntax and use

The following sections describe the syntax of Encina administrative commands, how to issue the commands interactively, and how to get online help. The examples use **tkadmin** commands to illustrate syntax. However, the command syntax is the same for other Encina command suites—the **rqsadmin**, **sfsadmin**, **ppcadmin**, and **drpcadmin** command suites.

Command syntax

All Encina administrative commands use the same general syntax conventions. The following shows these conventions:

```
suite verb object {argument...} -option  argument [-option] {argument | argument}
```

A description of each element of the command follows:

- **suite**—The command suite name specifies the group of related commands to which the command belongs.
- **verb object**—Two words describing the action initiated by the command and the object of that action. Some commands have verbs only.
- *argument*—Parameters that provide the command with information it uses to execute.

- **option**—A word preceded by a - (dash) that either tells the command to execute in a certain way or tells the command that a certain type of information follows in the form of an *argument*.
- { | } (braces containing a vertical bar) indicate that you can enter only one of the arguments separated by the vertical bars.
- {...} (braces containing an ellipsis) indicate that you can enter one or more instances of the argument or group of arguments that precede the ellipsis.
- [ ] (square brackets) indicate optional parts of the command.

The following syntax for the **tkadmin backup lvol** command illustrates these elements:

```
tkadmin backup lvol -server server_name logical_volume_name\
[-fileprefix file_prefix] [-filesize file_size] \
[-nfiles number_of_files]
```

In the preceding **tkadmin** example, **tkadmin** is the suite, **backup** is the verb, **lvol** is the object, **-server**, **-fileprefix**, **-filesize**, and **-nfiles** are options, and *server_name*, *logical_volume_name*, *file_prefix*, *file_size*, and *number_of_files* are arguments. The **-fileprefix**, **-filesize**, and **-nfiles** options are optional.

Using the Commands

When entering command names use a space to separate each element on a command line. Also use spaces to separate any multiple arguments. Do not use a space to separate an option from its - (dash).

Specifying units of size: Encina administration often involves managing physical storage. The unit of physical storage can be bytes, pages, or megabytes, depending upon the argument specification for individual commands. (Check the reference page of the appropriate command for a precise definition of the argument.) To make specification of very large numbers easier, Encina administrative commands accept a **k** or **K** appended to an integer argument, interpreting the **k** or **K** to mean a multiple of 1024. For example, **2k** is interpreted as 2048.

Using commands in interactive mode: Encina provides a shortcut for typing administrative commands interactively. You can omit the command suite name and type only the verb, object, and arguments. To enter commands interactively, enter the command suite name with no arguments—for example, **tkadmin**. When you do so, the prompt becomes the name of the command suite as the following example shows:

```
% tkadmin
Encina Toolkit administration tool.
Type "help" for help, "exit" to exit.
tkadmin>
```

Abbreviating commands: You can abbreviate a command name to the shortest form that still distinguishes it from other command names. You cannot, however, abbreviate the command suite name. For example, you can shorten the **tkadmin expand pvol** command to **tkadmin e p** because no other command in the **tkadmin** interface has a verb beginning with the letter "e" and a object beginning with the letter "p."

When two commands are similar, be sure to avoid ambiguous abbreviation. Consider the following commands: **tkadmin create logfile** and **tkadmin create lvol**. To avoid ambiguity, you can abbreviate these commands to **tkadmin c lo** and **tkadmin c lv**. Each letter of the final word of the command must be spelled out until you reach a character that distinguishes that command from all others. In this example, the last letter of the first command ("o") distinguishes it from the second command, which has a "v" in that character position.

Specifying the server name: Each Encina administrative command must indicate the server for which the command is being issued. The server can be specified by using the **-server** option or by using a server environment variable. A value provided on the command line overrides a value defined in an environment variable. The command examples in this document assume that the server environment variable has been set to the name of the server. The **-server** option is omitted on the command line.

Each server has its own server environment variable. The server environment variables are as follows:

- The ENCINA_TK_SERVER environment variable sets the name of the server for **tkadmin** commands.
- The ENCINA_RQS_SERVER environment variable sets the name of the RQS server for **rqsadmin** commands.
- The ENCINA_SFS_SERVER environment variable sets the name of the SFS server for **sfsadmin** commands.
- The ENCINA_GWY_SERVER environment variable sets the name of the PPC Gateway server for **ppcadmin** commands.
- The DRPC_GATEWAY environment variable sets the name of the DE-Light Gateway server for **drpcadmin** commands.

For administrators who frequently configure one Encina server, setting the server environment variables can conveniently reduce the amount of information that must be specified on the command line. The following steps describe the commands that are used to set environment variables on different operating systems.

On UNIX. To set a server environment variable in a C shell, use the **setenv** command. The following example sets the ENCINA_TK_SERVER variable to the server named **/.:/branch1/server/rqs1**:

```
% setenv ENCINA_TK_SERVER /.:/branch1/server/rqs1
```

On Windows NT. To set a server environment variable, use either of the following methods:

- At the DOS prompt, use the **set** command to set the variable. The following example sets the ENCINA_SFS_SERVER variable to the server named **/.:/branch1/server/sfs1**:

  ```
  C:\ set ENCINA_SFS_SERVER=/.:/branch1/server/sfs1
  ```

  Note that the value of the environment variable remains in effect only for commands issued in that DOS shell.
- From the **Start** menu, choose the **Settings** menu and then choose **Control Panel**. Double-click the **System** icon. Type the name of the server environment variable and its value in the text fields at the bottom of the dialog box. Choose the **Set** button to set the variable. The new environment variable becomes available the next time you log on.

Getting online help

You can receive help on any command that is part of a command suite by typing the name of the command suite followed by the command verb **help**. Three types of help are available:

- A complete list of commands in the command suite. For example, to display the list of **tkadmin** commands, enter the following:

  ```
  % tkadmin help
  ```
- The syntax of a specific command. For example, to display the syntax of a particular command, specify that command as the argument to the **tkadmin help** command, as follows:

  ```
  % tkadmin help query backup
  ```
- A list of all commands in the command suite that contain a specific keyword. For example, to display a list of all commands in the **tkadmin** command suite that contain the word **set**, specify **set** as the argument to the **tkadmin help** command, as follows:

  ```
  % tkadmin help set

  The commands containing the word "set" are
  tkadmin set minprotectlevel -- Set the server minimum protection
  level.
  tkadmin set authorization -- Set the server authorization level.
  tkadmin set trpctrantimeout -- Specify TRPC transaction timeout.
  ```

Using the enccp interface

The Encina Control Program (**enccp**) is a command-line and scripting interface for administering Encina Monitor cells. It provides all the functionality of Enconsole. The **enccp** interface is modeled after **dcecp**, the DCE administration tool based on TCL (tool command language). TCL is a portable command language that provides programming facilities, such as variables, procedures, conditionals, list-processing functions, and looping constructs. The **enccp** interface extends TCL by providing a set of commands for manipulating Encina Monitor objects. Furthermore, **enccp** incorporates **dcecp** so that all **dcecp** functionality is available from within **enccp**. See the *Encina Administration Guide Volume 3: Advanced Administration* and the **enccp** reference pages for details.

Using the DCE interface

By default, Enconsole creates the principals, groups, and keytab files needed for your servers. You can use Enconsole to view or change these settings. Enconsole also creates the access control lists (ACLs) that grant permissions to DCE resources. Additional DCE tasks involve setting up or modifying ACLs to grant access to Encina resources (such as granting clients permissions to Encina servers). Depending on your version of DCE, use either the **dcecp** or the **acl_edit** interface to modify ACLs. See your DCE documentation and "Chapter 8. Understanding and controlling access to Encina resources" on page 153 for more information.

Using the Tivoli Global Enterprise Manager

Tivoli provides a set of software tools for managing heterogeneous hardware and software resources within a network. GEM adds to the Tivoli environment the ability to identify and organize resources according to their business application and function. GEM, also, provides graphical views that administrators can use to inspect the overall status of distributed applications, to monitor individual resources within the application, and to perform administrative operations on resources.

GEM can be used in a network or a mainframe environment. In a distributed network environment, the Tivoli Management Framework must be configured. In a mainframe or OS/390 environment, NetView™ must be configured.

GEM provides the following functionality:

- Provides tools for modeling, monitoring, and administering cross-platform, distributed applications using graphical views
- Organizes multiple views into a hierarchical structure
- Reports status information about individual software components within a business system

- Provides controls for performing operations on individual resources
- Simplifies setting alerts and configuring event triggers on heterogeneous network resources through a uniform interface

To use GEM to monitor and manage Encina Monitor cells, several Tivoli components must be installed and configured. See the *Encina Administration Guide Volume 3: Advanced Administration* for a list of software requirements.

For complete information about Tivoli, refer to Tivoli documentation. For more information about GEM, see the *Tivoli Global Enterprise Manager Installation and User's Guide.*

Chapter 2. Administering servers in a Monitor cell

This chapter describes the basic administrative tasks you perform after you configure and start a Monitor cell. This chapter describes how to stop, restart, and manage a cell manager and node manager, and how to define, start, stop, and restart different types of servers. It also describes how to delete objects such as servers.

Restarting Enconsole

This section describes the procedures used for starting Enconsole to administer an Monitor cell that is already running. The procedure you use depends on your platform.

After successfully restarting Enconsole, you can log into the Distributed Computing Environment (DCE) as the Encina administrative principal (as described in *Planning and Installation Guide*) and perform maintenance tasks with Enconsole. Note that you can run more than one instance of Enconsole within a single Monitor cell at the same time, but you cannot change Monitor cells within a single Enconsole session.

On UNIX Systems. Enter the following command to start Enconsole with a Monitor cell that is already running:

```
% enconsole monitor_cell
```

Specify the cell-relative DCE name of an existing Monitor cell for the *monitor_cell* argument. For example, enter the following command to start Enconsole with a Monitor cell named **/.:/branch1**:

```
% enconsole  /.:/branch1
```

Enconsole opens the Enconsole Servers display screen and main menu.

On Windows NT Systems. Perform the following steps to start Enconsole:

Note: Do not run Enconsole from a network drive, RAM disk, or removable media.

1. Launch Enconsole, for example by choosing Enconsole from the **Start** menu.

Figure 15. Enconsole icon

Enconsole displays the Cell Name form.

2. Type the cell-relative DCE name of an existing Monitor cell (for example, **/.:/branch1**) in the **Cell Name** field and choose the **OK** button. Enconsole opens the Enconsole Servers display screen.

Stopping cell managers and node managers

This section describes how to stop cell managers and node managers by using either shell scripts (on UNIX systems) or the Enconsole interface.

Stopping node managers

Before you stop a node manager, you must stop any servers that are managed by the node manager. The following sections describe how to use Enconsole or a shell script to stop a node manager.

Note: If the state of the node manager is `Lost Contact` (this is shown on the Nodes display screen), the cell manager cannot communicate with the node manager. You cannot use either a shell script or Enconsole to stop the node manager in this case. Instead, you must log into the machine on which the node manager is running and use the operating system **kill** command (on UNIX systems) or the Task Manager (on Windows NT systems) to stop the node manager (**enm**) process.

Stopping node managers by using a shell script (UNIX systems only)

The **rc.encina.***node_name* shell script can be used to stop a running node manager. The script is stored in the node manager's working directory (*directory_root*/*monitor_cell*/**node/***node_name* by default, where *directory_root* is the name of your Encina directory root, *monitor_cell* is the name of your Monitor cell, and *node_name* is the name of the node manager you want to stop). You must be logged into DCE as the Encina administrative principal to execute this shell script.

To stop a running node manager, enter the following command at the prompt:

```
% rc.encina.node_name stop -all
```

The **-all** option tells Enconsole to shut down any servers running on the node before it shuts down the node manager. If you have running servers and you do not use the **-all** option, the script fails and the node manager is not shut down.

Stopping node managers by using Enconsole

While in Enconsole, perform the following steps to stop a running node manager:

1. Stop all servers running on the node. Use the procedure described in "Stopping servers" on page 64.
2. Choose the **Stop Node** command from the **Actions** menu, and select the node manager that you want to stop. Choose the **OK** button. Enconsole displays the Command Status display screen and stops the node manager.
3. Choose the **OK** button to close the Command Status display screen.

Stopping cell managers

Before you stop a cell manager, you must stop all servers that are running in the Monitor cell. Note that you do not need to stop the node managers in the cell before stopping the cell manager. You need to stop only the servers. The following sections describe how to use Enconsole or a shell script to stop a cell manager.

Stopping cell managers by using a shell script (UNIX systems only)

The **rc.encina.cell** shell script can be used to stop a running cell manager. It is stored in the cell manager's working directory (by default, *directory_root*/*monitor_cell*/**ecm**, where *directory_root* is the name of your Encina directory root and *monitor_cell* is your Monitor cell name). You must be logged into DCE as the Encina administrative principal to execute this shell script.

To stop a running cell manager, enter the following command at the prompt:

```
% rc.encina.cell stop -all
```

The **-all** option causes the script to shut down any servers still running in the Monitor cell before it shuts down the cell manager. If you have running servers and you do not include the **-all** option, the script fails and the cell manager is not shut down.

Stopping cell managers by using Enconsole

While in Enconsole, perform the following steps to stop a cell manager:

1. Stop all servers running in the Monitor cell. Use the procedure described in "Stopping servers" on page 64.
2. Choose the **Stop Cell** command from the **Actions** menu. Enconsole displays the Command Status display screen and stops the cell manager.
3. Choose the **OK** button to close the Command Status display screen.

Restarting cell managers and node managers

This section describes how to restart cell managers and node managers. You can use shell scripts (on UNIX systems) or command files (on Windows NT systems) to restart cell managers and node managers. When you use a shell script or command file, the cell manager or node manager process is monitored and restarted when necessary by the **encinaNanny** process. The **encinaNanny** process is started automatically by the script or command file.

On UNIX systems, the shell scripts are normally run by a master initialization file when the machine boots up. You can restart cell managers and node managers by using either of the following methods:

- Running shell scripts from an initialization file (recommended). See "Using an initialization file (UNIX systems)".
- Executing shell scripts from the command line. See "Executing shell scripts from the command line (UNIX systems)".

On Windows NT systems, the command files are normally run by the Encina Restart Service, which is configured using the **encinaRestartAdmin** program. You can restart cell managers and node managers by doing either of the following:

- Using the Encina Restart Service (recommended). This service automatically restarts cell managers and node managers after a machine outage and makes sure that they continue to run after the user who started them logs out. Once restarted, the node manager automatically restarts the remaining Encina servers on the machine as necessary. See "Using the Encina Restart Service (Windows NT systems)" on page 36.
- Executing command files from the command line. See "Using command files from the command line (Windows NT systems)" on page 39.

Using an initialization file (UNIX systems)

You can restart cell and node managers by running shell restart scripts from an initialization file when the machine boots up. The master initialization file can be edited to include the Encina cell and node manager restart scripts. The name of the initialization file is dependent on the operating system you are using. For example, the master file on Solaris systems is located in the **/etc/init.d** directory. On AIX systems, the master initialization file is **/etc/rc**.

Executing shell scripts from the command line (UNIX systems)

This section describes how to restart cell and node managers by executing shell restart scripts from the command line.

Executing the cell manager restart script

Before you attempt to restart a cell manager, the cell manager process (**ecm**) must be stopped. Execute the **rc.encina.cell** shell script that is stored in the cell manager's working directory (by default, *directory_root*/*monitor_cell*/**ecm**, where *directory_root* is the name of your Encina directory root and *monitor_cell* is your Monitor cell name). For example, enter the following command to restart the cell manager in a Monitor cell named **branch1**:

```
% /opt/encinalocal/branch1/ecm/rc.encina.cell
```

The cell manager is restarted. Note that if you stop the cell manager process with an operating system command such as **kill** or **kill -1**, the process exits successfully but is not restarted. If you kill the process by using the **kill -9** command, the script restarts it.

Note: You can customize the **rc.encina.cell** shell script to meet your needs. See your operating system documentation for more information about using shell scripts to automatically restart processes.

Executing the node manager restart script

Before you attempt to restart a node manager, the cell manager must be running and the node manager process (**enm**) must be stopped. Execute the **rc.encina.***node_name* script that is stored in the node manager's working directory (by default, *directory_root*/*monitor_cell*/**node/***node_name*, where *directory_root* is the name of your Encina directory root, *monitor_cell* is the name of your Monitor cell, and *node_name* is the name of the node manager you want to start). The partial syntax is as follows:

```
rc.encina.node_name [start] [-enable]
```

For example, enter the following command to restart a node manager named **machine1** in a Monitor cell named **branch1**:

```
# /opt/encinalocal/branch1/node/machine1/rc.encina.machine1 start -enable
```

The node manager named **machine1** is restarted. The example uses the **-enable** option with the **start** argument to first enable the node manager and then start it. (The **start** option is the default argument and need not be specified. It is shown here for completeness.) The **-enable** option sets the node manager's desired state to `Running`. This option is required if the node was stopped using either the script or Enconsole. For a complete description of the syntax for this script, see the online reference page.

Although the simple restart command (**rc.encina.***node_name* **start**) does not require any DCE credentials, you must log into DCE as a member of the group **encina_operator_group** or the group **encina_admin_group** to use the **-enable** option.

Note: Do not attempt to use the **-enable** option when running the restart script from a master initialization file.

Note that if you stop the node manager process with an operating system command such as **kill** or **kill -1**, the process exits successfully but is not restarted. If you kill the process with the **kill -9** command, the script restarts it.

Note: You can customize the **rc.encina.***node_name* shell script to meet your needs. See your operating system documentation for more information about using shell scripts to automatically restart processes.

Using the Encina Restart Service (Windows NT systems)

This section describes the Encina Restart Service. The topics covered included the following:

- Configuring the Encina Restart Service
- Changing the default account used to run the Restart Service
- Disabling and removing the Restart Service

Configuring the Encina Restart Service

Perform the following steps to enable automatic restart of a cell manager or node manager. The procedure assumes that you have already configured and started a cell manager and node manager by using Enconsole.

1. Run the **encinaRestartAdmin** program, located in the *installation_directory***\bin** directory, where *installation_directory* is the directory where Encina is installed (by default, **C: \opt\encina\bin**). The Encina Restart Service Administration window appears.
2. Choose the **Configure** button. The Configure Encina Restart Service window appears. Make sure that the pathname for the **encinaRestart.exe** file is shown in the **Executable** field. The default pathname is *installation_directory***\bin\encinaRestart.exe**, where *installation_directory* is the directory where Encina is installed. Change this value if necessary, and then choose the **OK** button. The Configure Encina Restart Service window exits and the Encina Restart Service is enabled.
3. Choose the **Add** button. The Edit Server window appears.
4. In the **Server name** field, enter a descriptive name for the cell manager to be restarted—for example, **cell2-startup**. This name is used to identify the cell manager restart process; it does not need to be associated with the cell manager or cell name.
5. In the **Command line** field, enter the pathname of the server's ***.cmd** file, located in the server's working directory. The default pathname of the cell manager's restart command file is

*directory_root**monitor_cell***ecm****encina.cell.cmd**, where *directory_root* is the name of the Encina directory root and *monitor_cell* is the name of your Monitor cell.

6. In the **Directory** field, enter the pathname of the working directory for the cell manager. By default, this is the directory in which the restart command file is stored.
7. Choose the **OK** button. The Edit Server window exits and the server's restart configuration is displayed in the Encina Restart Service Administration window. Note that you can change a server's restart configuration by selecting it from the list and choosing the **Modify** button or double-clicking the name of the server in the list.
8. Repeat Steps 3 through 7, specifying information for the node manager. The default pathname of the node manager's restart command file is *directory_root**monitor_cell***node***node_name***encina.***node_name***.cmd**, where *directory_root* is the name of the Encina directory root, *monitor_cell* is the name of your Monitor cell, and *node_name* is the name of the node.
9. Choose the **Close** button to exit the Encina Restart Service Administration window.
10. Restart the machine to enable the Encina Restart Service. The cell manager and node manager restart automatically and continue to run when you log out.

    Note: The cell and node manager must be running when you log out or restart the machine. If they are stopped when you log out or reboot, the Restart Service will not restart them.

When using the Encina Restart Service, note the following:

- When the Encina Restart Service is enabled, Encina stores shared memory segments in the **%TMPDIR%** directory by default. If this directory does not already exist, create it or set the ENCINA_SHM_DIR environment variable to the full pathname of another directory.
- When using the command files to restart Encina processes (whether through the Encina Restart Service or from the command line), disable automatic ("application exception" or "just-in-time") debugger invocation. To disable automatic debugger invocation, make sure that the ENCINA_ALLOW_DEBUGGING environment variable is not set in the Control Panel/System Properties window or in Enconsole.

  Any Encina process that must be available continuously must not have this environment variable set. If you do not disable automatic debugging, a debugger is started automatically when a process generates an unhandled exception. The debugger can fail to propagate an error exit status from the process being debugged and can also prevent the process from exiting until the debugger window is dismissed. This can prevent the command files from restarting Encina processes. Set the ENCINA_ALLOW_DEBUGGING

environment variable only for processes that are being debugged or are running in an application development environment.

Changing the default account used by the Restart Service

By default, the Encina Restart Service uses the special internal system account. This account is used by the operating system and by services running under Windows NT. The system account has the advanced right ″Log on as a service.″ If you want to run the restart service under another user account, that account must be granted the ″Log on as a service″ right. By default, only the account under which the Encina Restart Service runs can use the Task Manager to terminate the cell and node manager processes. However, using the Task Manager to terminate a process can cause undesired results including loss of data and system instability.

To run the Restart Service with an account other than the system account, perform the following steps:

1. Start the User Manager.
2. From the **Policies** menu, choose **User Rights**. In the User Rights Policy dialog box, scroll through the **Right** list and choose ″Log on as a service.″ (Note that this user right is not automatically displayed in the list. You must check the **Show Advanced User Rights** checkbox at the bottom of the dialog box.) The **Grant To** list displays users and groups that hold the right shown in the **Right** field.
3. Choose the **Add** button. The Add Users and Groups dialog box appears.
4. In the **List Names From** field, choose from the list of workstation and domain names. The **Names** list displays the groups and users of the domain or workstation selected. (Choose the **Show Users** button to display user names.)
5. Choose a user or group from the **Names** list. Choose the **Add** button. The **Add Names** list displays the groups or users to be granted the selected right.
6. Choose the **OK** button to close the Add Users and Groups dialog box. Choose the **OK** button to close the User Rights Policy box. Exit the User Manager.
7. Run the **encinaRestartAdmin** program, located in the *installation_directory***bin** directory, where *installation_directory* is the directory where Encina is installed (by default, **C:\opt\encina\bin**). The Encina Restart Service Administration window appears.
8. Choose the **Configure** button. The Configure Encina Restart Service window appears.
9. In the Identity box, deselect the **Use system account** checkbox.
10. Enter a username and password in the **UserName** and **UserPassword** fields. The username must be specified in the form *domain**user_name*.

11. Continue configuring the Restart Service.

Disabling and removing the Restart Service

Disabling the Encina Restart Service prevents it from being used by a user or a dependent service. (The service can later be enabled for automatic or manual start.) Removing the Restart Service deletes it from the Windows NT Registry. You must reconfigure the service to use it again.

To disable the Encina Restart Service, use the same procedure that you use to disable any other service. You can disable the service either from the Control Panel/Services window or from the Configure Encina Restart Service window. In the Configure Encina Restart Service window, choose the Disabled option from the Startup list.

To remove the Encina Restart Service, choose the **Remove** button on the Encina Restart Service Administration window.

Using command files from the command line (Windows NT systems)

This section describes how to restart a cell manager and node manager by executing command files from the command line.

Executing the cell manager command file

Before you attempt to restart a cell manager, the cell manager process (**ecm**) must be stopped. Execute the **encina.cell.cmd** command file that is stored in the cell manager's working directory (by default, *directory_root**monitor_cell***ecm**, where *directory_root* is the Encina directory root and *monitor_cell* is your Monitor cell name). For example, enter the following command to restart the cell manager in a Monitor cell named **branch1**:

```
C:\> \opt\encinalocal\branch1\ecm\encina.cell.cmd
```

The **encina.cell.cmd** command file starts the cell manager. After the cell manager restarts, you can restart Enconsole (as described in "Restarting Enconsole" on page 31) and perform other administrative tasks.

Executing the node manager command file

Before you attempt to restart a node manager, the cell manager must be running and the node manager process (**enm**) must be stopped. Also, you may need to be logged into DCE as a member of the group **encina_operator_group** or the group **encina_admin_group**, depending on how the node manager was stopped. If you stopped the node manager by using Enconsole, you must have DCE credentials to execute the command file. If you stopped the node by some other method, DCE credentials are not

needed. For example, you do not need DCE credentials if a logout or machine reboot stopped the node manager (and you are not using the Encina Restart Service).

Execute the **encina.***node_name***.cmd** command file that is stored in the node manager's working directory (by default, *directory_root**monitor_cell***node***node_name*, where *directory_root* is the Encina directory root, *monitor_cell* is your Monitor cell name, and *node_name* is the node name). The script has two different syntaxes:

encina.*node_name***.cmd** [**start**] [**-enable**]

encina.*node_name***.cmd enable**

The **start** option starts the specified node manager. If you do not specify this argument, the script assumes you want to start a node manager. The **-enable** option sets the node manager's desired state to `Running`. The **enable** option is required if the node was stopped using Enconsole. Use the **enable** option with the **start** option to first enable the node manager and then start it.

The **enable** argument (specified as the only argument to the script) sets the node manager's desired state to `Running`. This argument simply enables the node manager. You must then re-execute the command file to start the node manager.

For example, enter the following command to restart a node manager named **machine1** in a Monitor cell named **branch1**:

```
C:\> \opt\encinalocal\branch1\node\machine1\encina.machine1.cmd start -enable
```

The node manager named **machine1** is restarted. The following sequence of commands does the same thing:

```
C:\> \opt\encinalocal\branch1\node\machine1\encina.machine1.cmd enable
C:\> \opt\encinalocal\branch1\node\machine1\encina.machine1.cmd start
```

Note: Do not attempt to use the **-enable** option or the **enable** argument when specifying the restart command file pathname in the Encina Restart Service.

After the node manager restarts, you can restart Enconsole (as described in "Restarting Enconsole" on page 31) and perform other administrative tasks.

Defining and starting servers

This section describes how to use Enconsole to define and start a server. The following steps summarize the procedure for defining and starting a server:

1. Set up the local environment on the node on which the server is to run. See *Planning and Installation Guide*.

2. Start Enconsole. See “Restarting Enconsole” on page 31.
3. Define and start a node manager. See *Planning and Installation Guide*.
4. Choose a server type. See “Choosing a server type”.
5. Define the server. See “Defining servers” on page 59.
6. Start the server. See “Starting servers” on page 63.

The following sections describe the different types of server definition forms and the procedures for defining and starting servers in more detail.

Choosing a server type

This section describes the server types that you can administer through Enconsole. Before you can define a server, you must choose the type of server you want to define. Figure 16 shows the Enconsole menu options used to define various server types. To choose a server type, choose the **Define Server** command from the **Actions** menu, and then select the type of server you want to define.

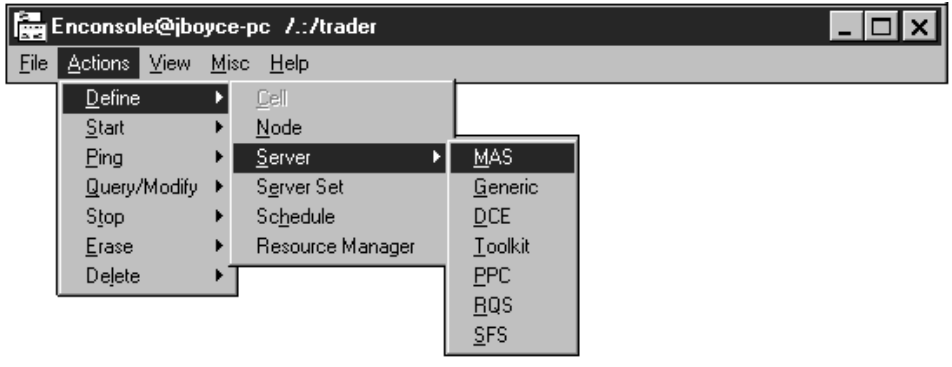

Figure 16. Define Server command

You can define the following types of servers:

- **MAS** — Monitor application servers (MAS); see “Monitor application servers” on page 45.
- **Generic** — Generic servers; see “Generic servers” on page 42.
- **DCE** — DCE application servers; see “DCE application servers” on page 43.
- **Toolkit** — Encina Toolkit servers, including DCE Encina Lightweight Client (DE-Light) Gateways; see “Encina Toolkit servers” on page 49. For information on DE-Light Gateways, see “DE-Light Gateways” on page 57.
- **PPC** — Peer-to-Peer Communications (PPC) Gateway servers; see “PPC Gateway servers” on page 51.
- **RQS** — Recoverable Queueing Service (RQS) servers; see “RQS servers” on page 53.

- **SFS** — Structured File Server (SFS) servers; see "SFS servers" on page 55.

Figure 17 shows the taxonomy of server types available via Enconsole.

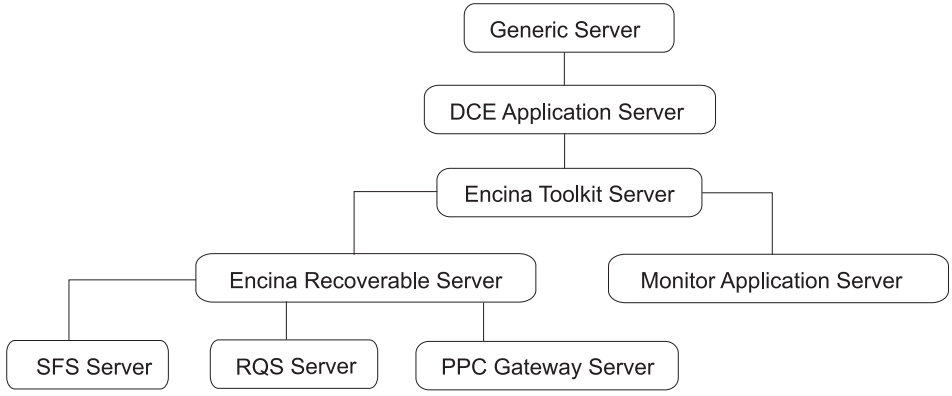

Figure 17. Server type hierarchy

Encina recoverable servers (SFS, RQS, and PPC Gateway) use the Encina Recovery Service to log transaction state information and changes to data. In the event of a media failure, the logged information can be replayed and applied to existing data to restore it to a consistent state. As shown in Figure 17, Monitor application servers, generic servers, and DCE application servers do not use the Encina Recovery Service to maintain persistent data.

Enconsole provides a specific server form for defining each server type. Each server form provides access to the server's operating system values through the Process Options form. Additionally, when the server type relies on DCE and Encina, you can access the Encina/DCE Options form to specify tracing options. All servers that are stored in your Monitor cell can also access the Monitoring Options form where you can specify ping options.

The following sections provide more information about the different types of servers and the forms used to define them. You can also use Enconsole online help to review information about these forms.

Generic servers

A generic server can be any server—for example, a shell program, a server written with the C programming language, or a server written with the X/Open TX application programming interface (API). The TX interface is described in the *Encina Toolkit Programming Guide*. Although a generic server can use DCE and Encina capabilities, all generic servers do not necessarily rely on the DCE libraries or the Encina Toolkit libraries. Additionally, it is more appropriate to use the DCE Server form to define DCE application

servers and the Encina Toolkit Server Form to define Encina Toolkit servers because Enconsole automatically provides the fields necessary to configure these types of servers on their specific server forms.

When you choose the **Generic** command from the **Server** menu, Enconsole displays the Generic Server form (shown in Figure 18). You can use this form to specify the server name, node name, executable pathname, and command-line arguments for a generic server.

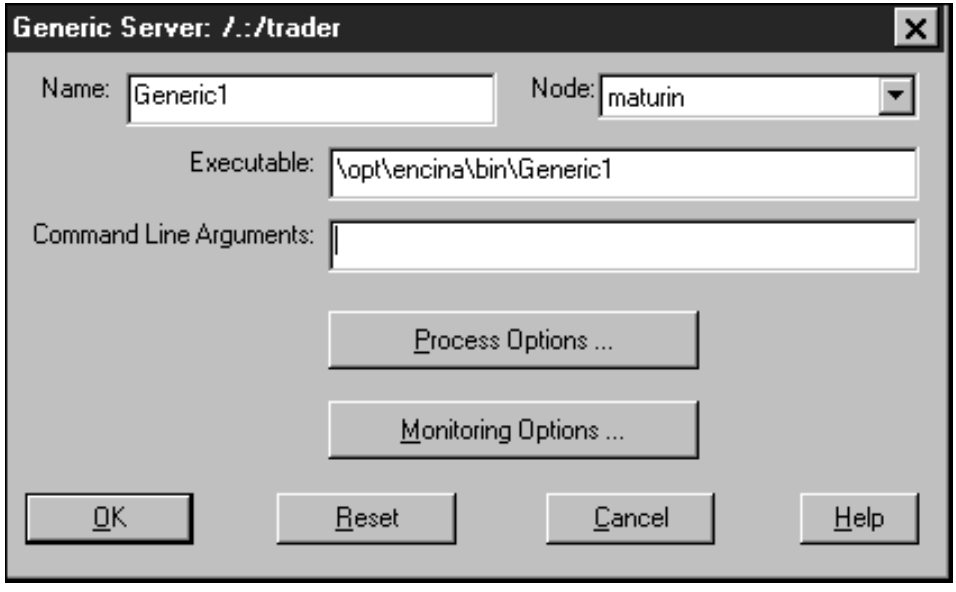

Figure 18. Generic Server form

DCE application servers

A DCE application server relies on DCE libraries. DCE application servers are typically written with the C programming language and the X/Open TX API. The DCE libraries are described in the *OSF DCE Application Development Guide* and the *OSF DCE Application Development Reference*, and the TX interface is described in the *Encina Toolkit Programming Guide*.

Each DCE application server can support distributed applications through remote procedure calls (RPCs) and through integrated system components such as the DCE Security Service. DCE application servers can also support portability across heterogeneous platforms and data sharing via the DCE Cell Directory Service (CDS).

When you choose the **DCE** command from the **Server** menu, Enconsole displays the DCE Server form (shown in Figure 19 on page 44). You can use

this form to specify the server name, node name, executable pathname, and command-line arguments for a DCE application server.

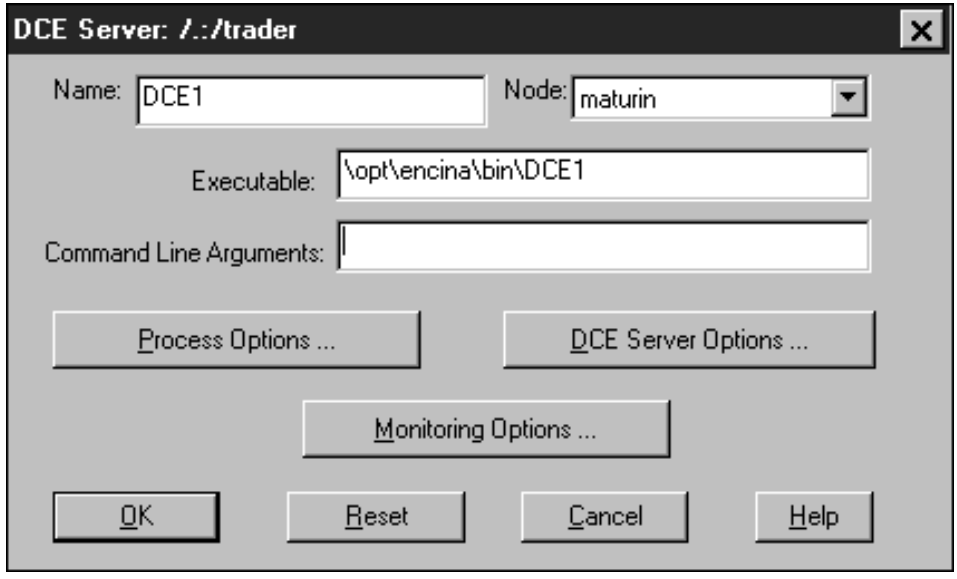

Figure 19. DCE Server form

Note: To administer a DCE application server with Encina, you must write the application so that it exports the object UUID to the endpoint database and CDS.

When you choose the **DCE Server Options** button from the DCE Server form, Enconsole displays the DCE Server Options form (shown in Figure 20 on page 45). You can use this form to specify additional attributes for a DCE server. Unless you are sure that you need to change these values, accept the default values.

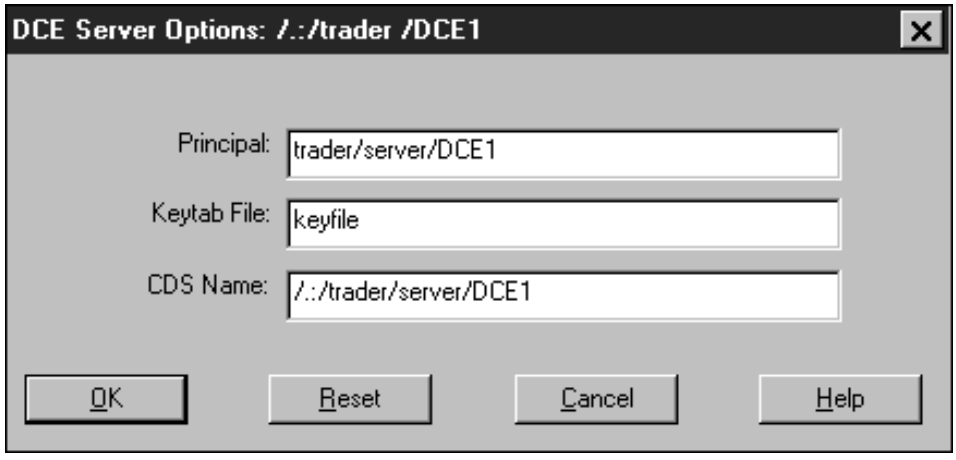

Figure 20. DCE Server Options form

Monitor application servers

A Monitor application server (MAS) processes client requests, usually by interacting with one or more resource managers that store application data. Each Monitor application server runs on a managed node in the Monitor cell. Each server can be replicated on its node and managed as a single entity. Replication makes multiple copies of the server available and helps ensure continued availability of the server even in the event of hardware or network problems. The Encina Monitor API is used to write Monitor application servers. This API is described in the *Encina Monitor Programming Guide* and in *Writing Encina Applications*.

When you choose **MAS** from the **Server** menu, Enconsole displays the Monitor Application Server form (shown in Figure 21 on page 46). This form is used to specify the server name, node name, executable pathname, command-line arguments, and interfaces for a Monitor application server. The interface names that you specify in the Interfaces field (by selecting the **Add** button and then entering the name of an interface) must match the names defined in the Monitor application server's Transactional Interface Definition Language (TIDL) file.

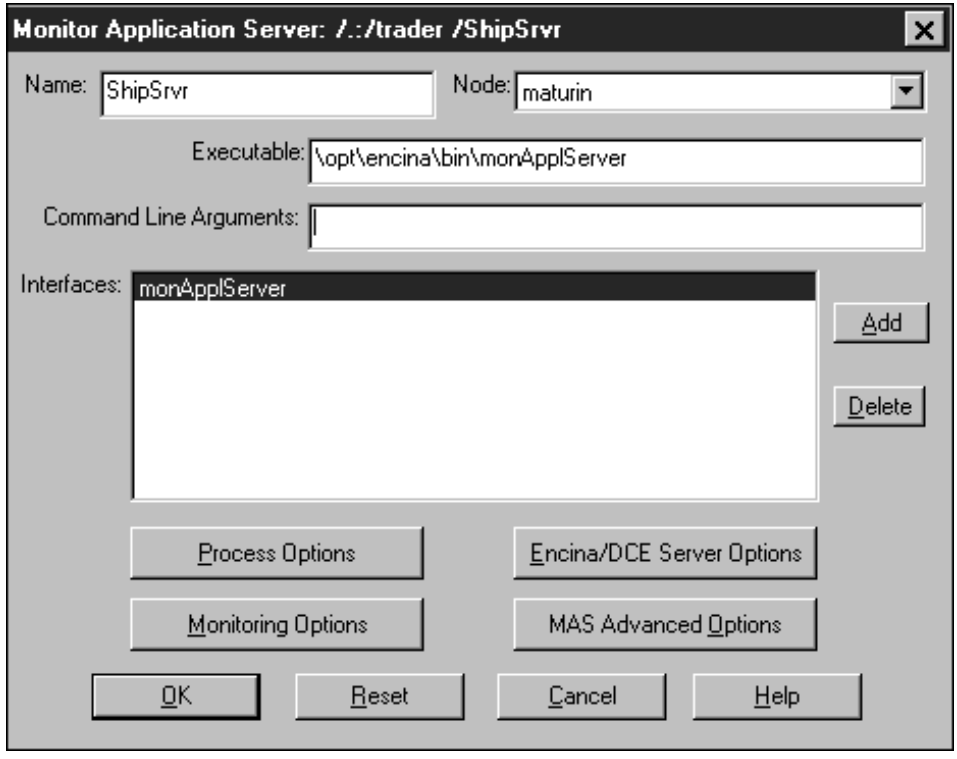

Figure 21. Monitor Application Server form

When you start a Monitor application server, the system registers the server's interfaces so that client requests can be routed to a server that exports the requested interface.

When you choose the **MAS Advanced Options** button on the Monitor Application Server form, Enconsole displays the MAS Advanced Options form. As shown in Figure 22 on page 47, you can use this form to specify attributes such as the number of processing agents (PAs) and their working directories. For PPC executive applications, you can also specify the PPC logical unit (LU) name.

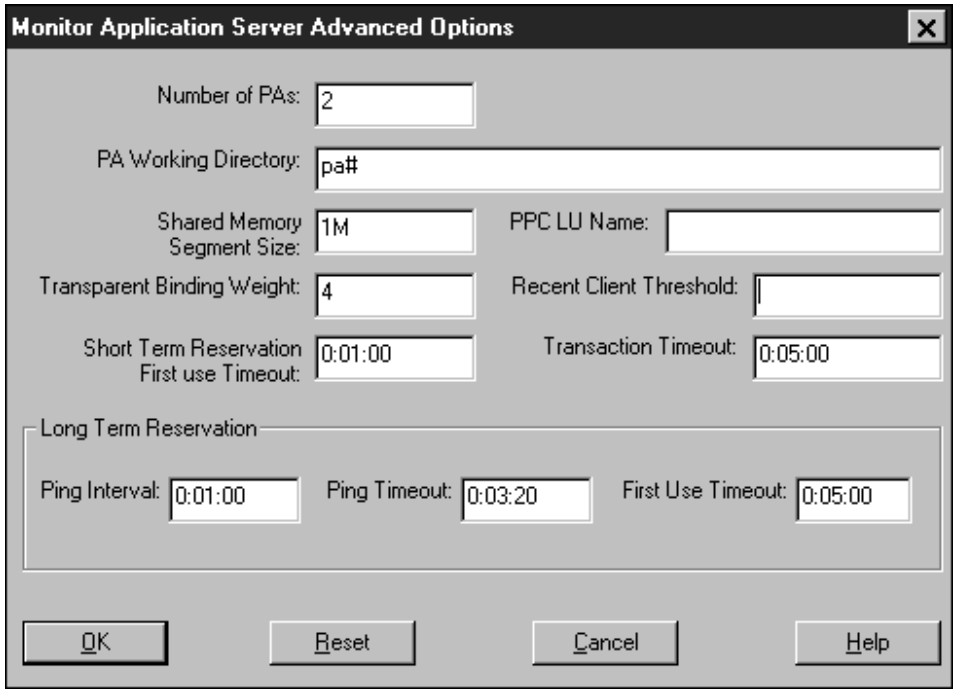

Figure 22. Monitor Application Server Advanced Options form

The following types of servers can also be run under the control of the Encina Monitor, and can therefore be configured and run as Monitor application servers:

1. Encina++/DCE servers.
2. Encina++/CORBA-DCE servers, which use Encina++ in a mixed Common Object Request Broker Architecture (CORBA) and DCE environment.
3. Specialized Monitor application servers (called bridge servers) that connect Java-based systems to Encina++/DCE systems, Encina/DCE systems, or both.

Encina++/DCE servers and Encina++/CORBA-DCE servers are configured and started in the same way as other Monitor application servers. However, the procedure for configuring and starting a bridge server in Enconsole is slightly different from the procedure for starting other Monitor application servers. The procedure includes steps for configuring the server to use the IBM C++ Object Request Broker (ORB) and the name service. To configure and start a bridge server, perform the following steps:

1. Define a new Monitor application server. On the Monitor Application Servers Form, specify values for the Server Name, Node Name, and Executable fields *only*. The name of the MAS bridge server must be the

same as the bridge server name specified in the **eomregimpl** command. This command is used to register the bridge server with the IBM C++ ORB.

2. The server's interfaces are registered with JNDI instead of with the DCE Directory Service. Therefore, instead of specifying interface names in the Interfaces field in Enconsole, you must define one home object name per interface to be exported to JNDI by the bridge server. To export a home object, establish a name binding for the object by defining an environment variable and setting its value to the name of the object. The Java client will use this name to find the home object in the namespace.

   The name of the environment variable must be of the form **ENCINA**_*moduleName_interfaceName*__Home__. Its value is any arbitrary value, as long as it is the same value used by the client to specify the home object. For example, if the interface name is **account** and the module name is **customers**, set the environment variable as follows:

   ```
   ENCINA_customers_accountHome  = mybank/accounts
   ```

   The environment variable must be set in Enconsole (it cannot be set at the operating system level). To set environment variables in Enconsole, choose the Process Options button on the Monitor Application Server form. Then choose the Environment Variables button. On the displayed form, specify a name and value and then choose the Set button.

3. Assign values to the following environment variables as part of the MAS definition: PATH, SOMCBENV, SOMCBDIR, SOMDGETENV, and NLSPATH. These variables define the location of IBM C++ ORB libraries and message catalogs, as well as other configuration information needed for the ORB. Note that the values for these variables *must* match the values specified for the same variables during bridge server development.

4. Assign values to the following environment variables as part of the MAS definition. These variables are used to specify which host and port the name service is running on.

   - ENCINA_CBORB_NS_HOST. This variable specifies the host name where the name service is executing (for example, **pc1.abc.com**). If this variable is not specified, the name service on the local host is used.
   - ENCINA_CBORB_NS_PORT. This variable specifies the port number on which the name service is listening. The default port number is 900. On UNIX systems, port number 900 is a reserved port. If that port is not being used, use this variable to specify the correct port number.

5. Set the Number of PAs field to **1** to complete the definition of the MAS. Note that only one PA is used for a bridge server. (Distribution of the workload among multiple PAs is not supported for calls from Java clients to bridge servers.)

See *Encina Object-Oriented Programming Guide* for more information on registering Encina++/CORBA-DCE and bridge servers with the appropriate ORB.

Encina Toolkit servers

An Encina Toolkit server is a server that uses the components of the Encina Toolkit to provide distributed transaction processing services. Encina Toolkit servers are multithreaded, allowing them to process several operations concurrently. Additionally, each Encina Toolkit server can store data in an XA-compliant resource manager (for example, a database) and access the data via the resource manager API (for example, Standard Query Language (SQL)) and the X/Open TX API. Encina recoverable servers (RQS, SFS, and PPC gateway) and Monitor application servers are Toolkit servers. DCE Encina Lightweight Client (DE-Light) Gateways are also Toolkit servers.

Encina Toolkit servers rely on the DCE services and the components of the Encina Toolkit Server Core. Typically, Encina Toolkit servers are built with Transactional-C (Tran-C). The Tran-C programming language simplifies the development of transaction processing applications by providing a high-level interface to the Encina Toolkit development tools. As a set of extensions to the C language, Tran-C also provides direct access to low-level constructs, such as threading, used by the Toolkit itself. Tran-C is described in the *Encina Transactional Programming Guide*. The Encina Toolkit Server Core is described in the *Encina Toolkit Programming Guide*.

When you choose the **Toolkit** command from the **Server** menu, Enconsole displays the Encina Toolkit Server form (shown in Figure 23 on page 50). You can use this form to specify Toolkit server attributes such as the server name, node name, executable pathname, and command-line arguments.

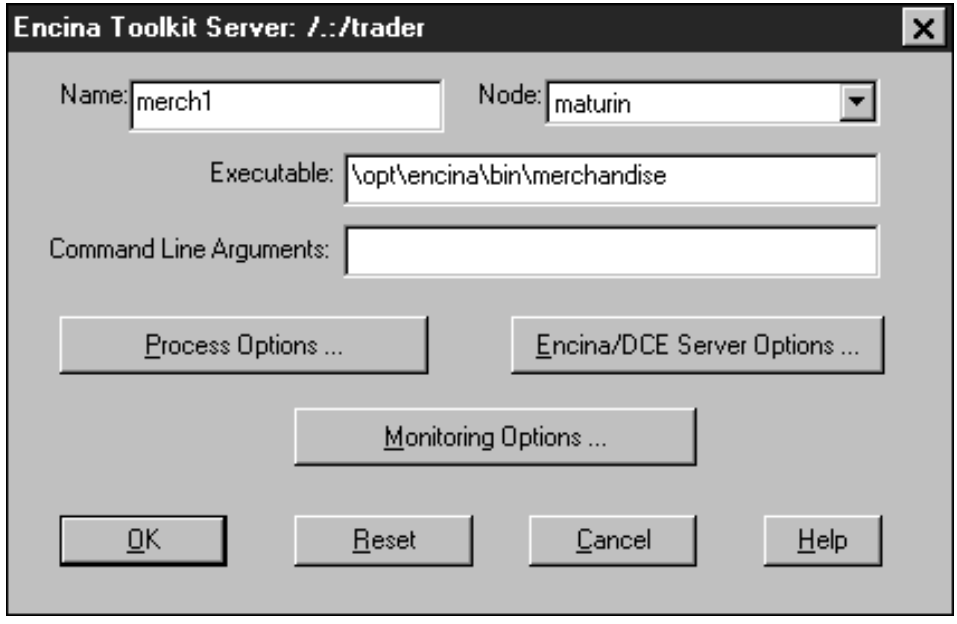

Figure 23. Encina Toolkit server form

When you choose the **Encina/DCE Server Options** button from the Encina Toolkit Server form, Enconsole displays the Encina/DCE Server Options form (shown in Figure 24 on page 51). You can use this form to specify additional DCE and Encina attributes for an Encina Toolkit or recoverable server.

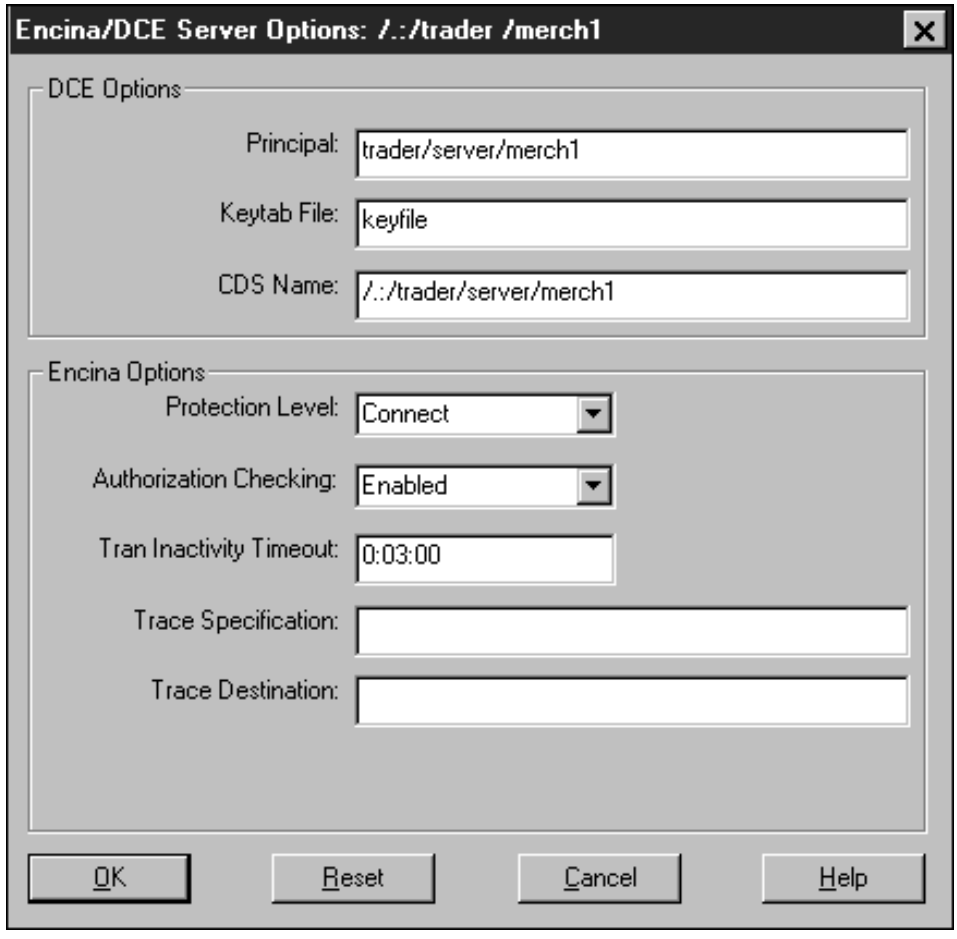

Figure 24. Encina/DCE Server Options form

The final two fields of the form define the server's trace options. If you leave the trace fields blank instead of specifying your own values, the system updates the trace fields with the default trace information for the server type. For more information about changing the trace values for a server, see "Chapter 10. Using the Encina Trace Facility" on page 211.

PPC Gateway servers

A Peer-to-Peer Communications (PPC) Gateway server is an Encina recoverable server that provides communications and distributed transaction processing capabilities between Encina and other systems. These other systems are typically mainframes that have System Network Architecture (SNA) LU 6.2 communications and transaction-management interfaces. By

using the PPC Services, Encina systems can request services from mainframe-based applications, and Encina systems can service requests made by mainframe applications.

PPC Executive applications communicate with a non-Encina system through the PPC gateway server. PPC Executive applications are written with the APIs described in the *Encina PPC Services Programming Guide*. To define a PPC Executive application through Enconsole, use the Monitor Application Server Advanced Options form, as described in "Monitor application servers" on page 45.

Note: Before you use Enconsole to define a PPC Gateway server, you must set up SNA as described in the *Encina Administration Guide Volume 2: Server Administration*.

When you choose the **PPC** command from the **Server** menu, Enconsole displays the PPC Gateway Server form (shown in Figure 25). The PPC Gateway Server form shows the PPC Gateway server attributes. You can use this form to assign the server name, node name, and disk names or logical volume names for a PPC Gateway server.

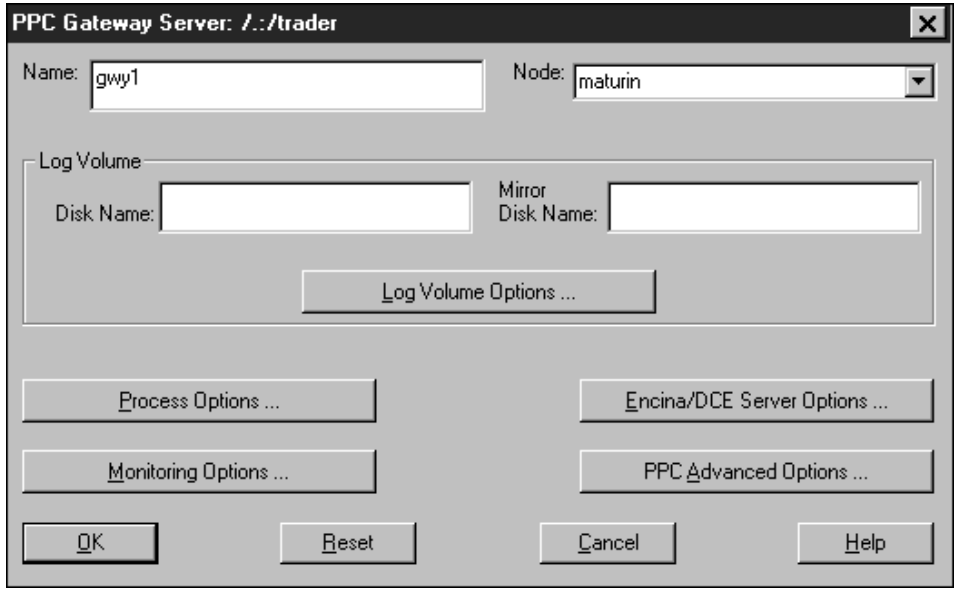

Figure 25. PPC Gateway Server form

When you choose the **PPC Advanced Options** button from the PPC Gateway Server form, Enconsole displays the PPC Gateway Server Advanced Options form (shown in Figure 26 on page 53). You can use this form to specify

additional attributes of a PPC Gateway server. Unless you are sure that you need to change these values, accept the default values.

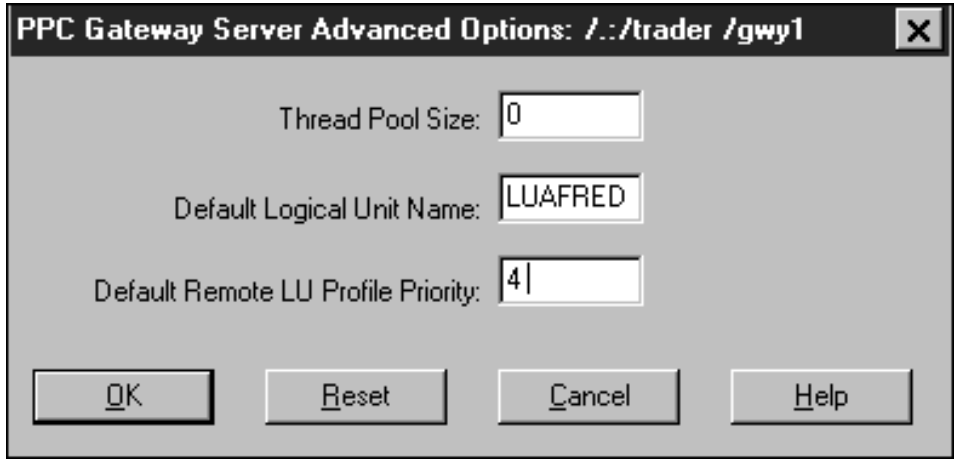

Figure 26. PPC Gateway Server Advanced Options form

RQS servers

A Recoverable Queueing Service (RQS) server is an Encina recoverable server that manages simultaneous requests for queued data. RQS servers support transactional client applications that must offload all or part of a task for later processing.

When you choose the **RQS** command from the **Server** menu, Enconsole displays the RQS form (shown in Figure 27 on page 54). The RQS form shows the RQS server attributes. You can use this form to assign the server name, node name, and disk names or logical volume names for an RQS server.

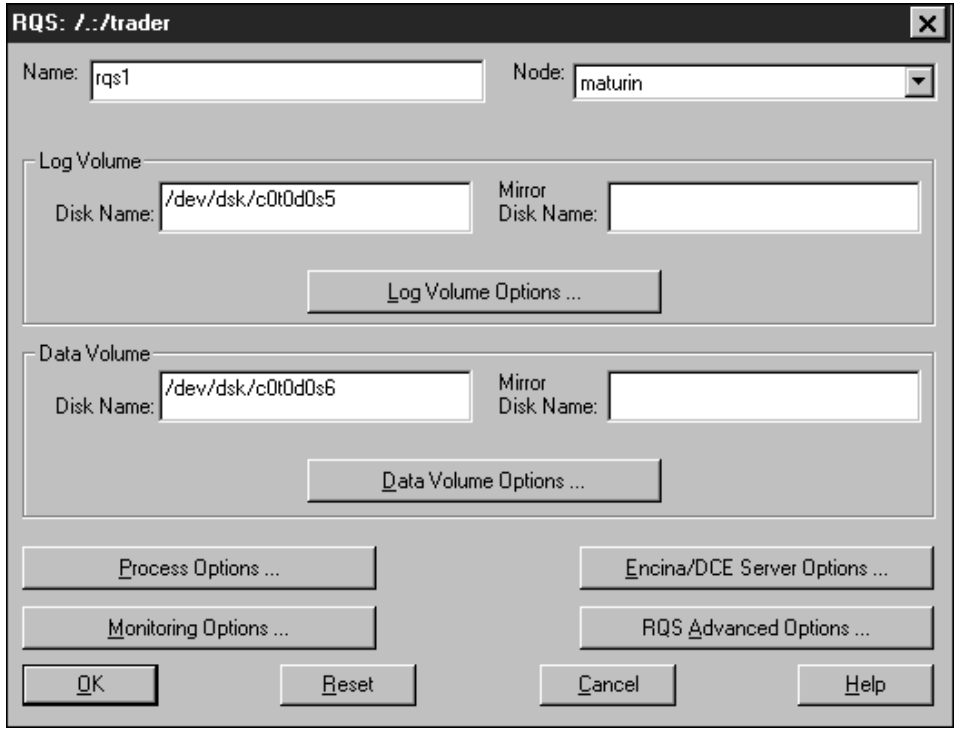

Figure 27. RQS form

When you choose the **RQS Advanced Options** button from the RQS form, Enconsole displays the RQS Advanced Options form (shown in Figure 28 on page 55). You can use this form to specify additional attributes of an RQS server. Unless you are sure that you need to change these values, accept the default values.

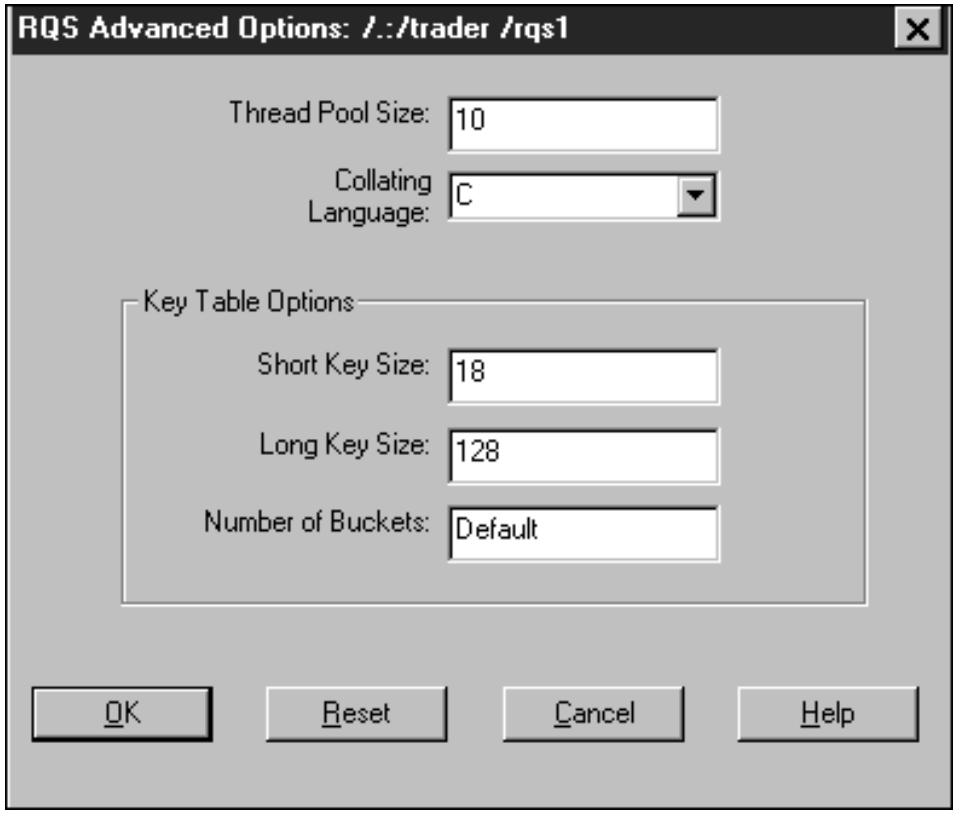

Figure 28. RQS Advanced Options form

SFS servers

A Structured File Server (SFS) is an Encina recoverable server that manages access to data stored in record-oriented files. It provides transactional integrity and flexible storage management. SFS servers are suited for applications that manage large amounts of data such as inventories, customer orders, and employee files.

When you choose the **SFS** command from the **Server** menu, Enconsole displays the SFS form (shown in Figure 29 on page 56). You can use this form to specify the server name, node name, and disk names or logical volume names for an SFS server.

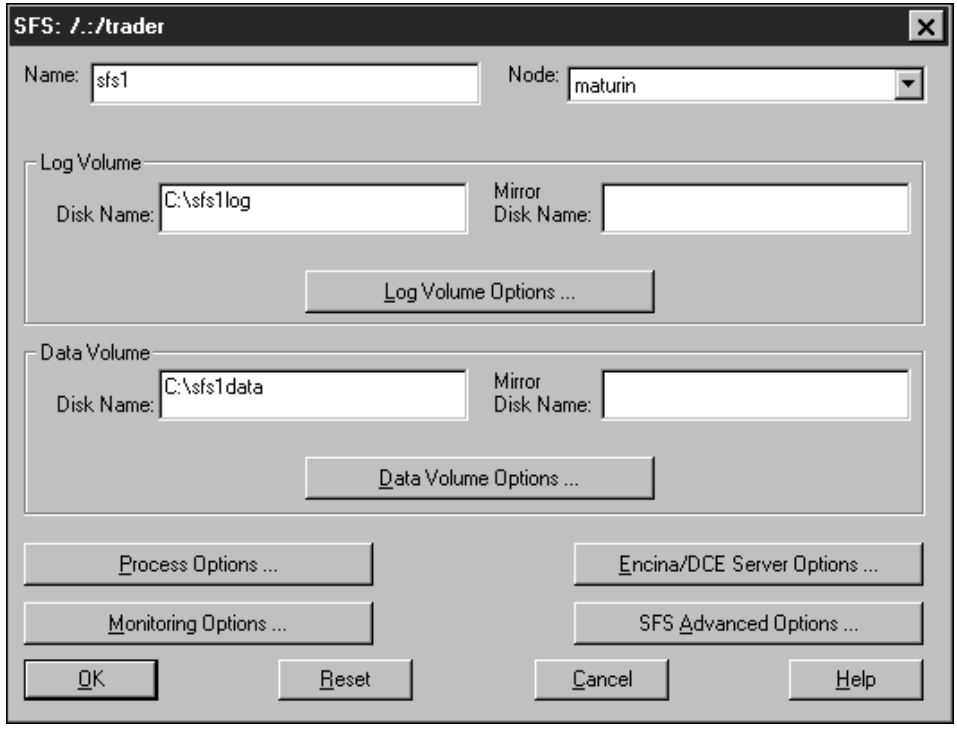

Figure 29. SFS form

When you choose the **SFS Advanced Options** button from the SFS form, Enconsole displays the SFS Advanced Options form (shown in Figure 30 on page 57). You can use this form to specify additional attributes of an SFS server such as the user thread pool size and emergency thread pool size. Unless you are sure that you need to change these values, accept the default values.

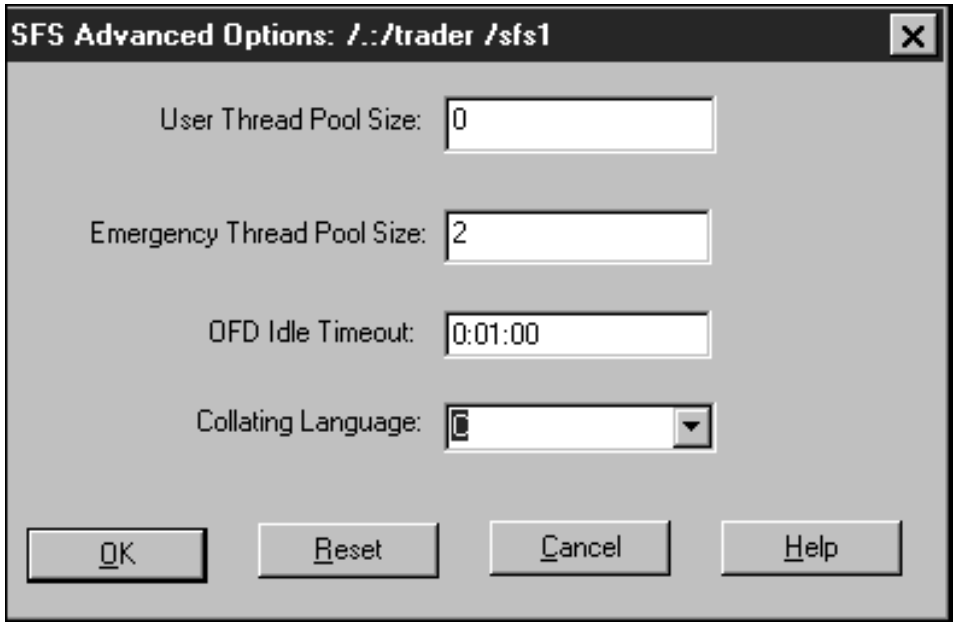

Figure 30. SFS Advanced Options form

DE-Light Gateways

A DCE Encina Lightweight Client (DE-Light) Gateway is a nonrecoverable Encina Toolkit server that enables communications between DE-Light clients and Encina or DCE servers. DE-Light clients, written in Java or C, enable systems that are not running as DCE clients to access the resources of Encina and DCE servers.

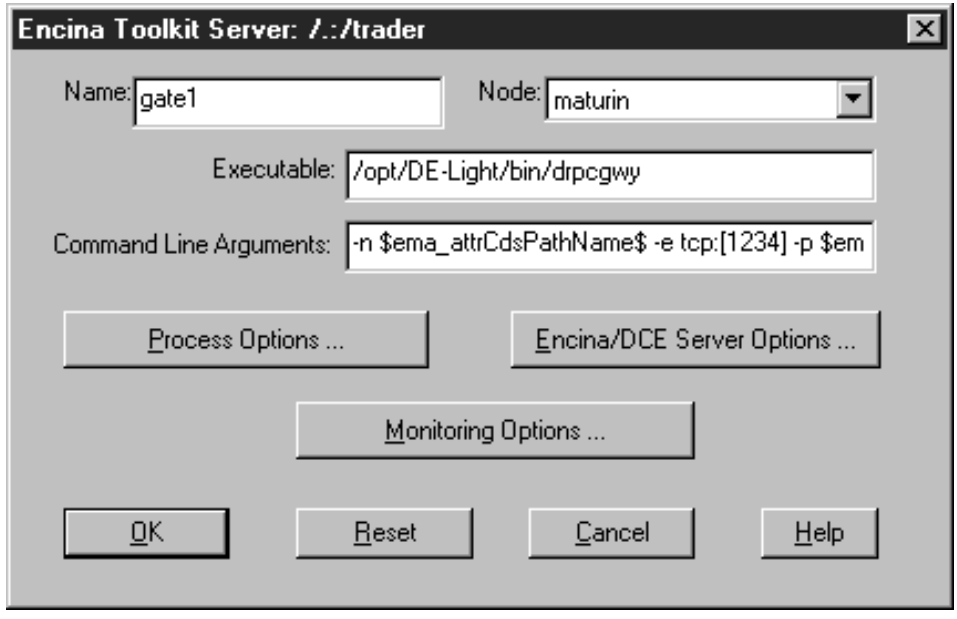

Figure 31. Defining a DE-Light Gateway from the Encina Toolkit server form

To define a DE-Light Gateway, select the **Toolkit** command from the **Server** menu and enter the following values into the specified fields of the Encina Toolkit server form. Figure 31 shows the Encina Toolkit Server form with the values required to define a DE-Light Gateway added to the fields.

Name *gateway_name*
: The value *gateway_name* is the name used in CDS to refer to your gateway (for example, **gate1**).

Node *node_name*
: The value *node_name* is the name of the node on which the gateway runs (for example, **mynode**).

Executable *install_dir***/bin/drpcgwy**
: The value *install_dir***/bin/drpcgwy** is the pathname for the gateway executable binary file.

Command Line Arguments
: Specify the following values for the various options to the **drpcgwy** command:

  -n $ema_attrCdsPathName$

  The **$ema_attrCdsPathName$** variable used with the **-n** option enables Enconsole to create an object in CDS for the gateway, using the name specified in the **Name** field.

  -p $ema_attrPrincipalId$

The **$ema_attrPrincipalId$** variable used with the **-p** option enables Enconsole to create a DCE principal for the gateway using standard Encina naming conventions.

-k $ema_attrKeyTabFile$

The **$ema_attrKeyTabFile$** variable enables Enconsole to create a keytab file for the principal, again using standard Encina naming conventions.

Any **drpcgwy** command options can be used in this field. For information on **drpcgwy** command options, see the *Encina Administration Guide Volume 3: Advanced Administration*.

-e tcp:[1234]

The required **-e** option specifies the transfer protocol identifier and endpoint that clients must use to communicate with the gateway. The protocol identifier can be either **tcp** (TCP/IP) or **http** (HTTP or HTTPS). If the protocol is HTTPS, you must specify two endpoints, nonsecure followed by secure, separated by a comma, within the brackets (for example, **http:[1234,2345]**). In this example, the protocol is TCP/IP, and the endpoint is port **1234**.

-A encina_admin

The **-A** option defines the exclusive administration principal. This principal is solely authorized to issued administrative commands to the gateway.

-Z 1

The **-Z** option enables exclusive authorization checking. With the **-Z** option set to **1**, the user is required to authenticate as the exclusive authorization principal before administering the gateway.

Defining servers

This section contains the procedure for defining a server with Enconsole. *To define and start a server, you can use Enconsole on any node.* To define a server, you specify all the information required to start the server and store the information under the server's name. The server definition procedure is similar for all server types; however, some server types require additional information specific to that type.

The login requirements for defining and starting a server are as follows:

- You must be logged into DCE as the Encina administrative principal (using the Enconsole interface).
- **For Servers to Be Run on UNIX Systems**. There are no local operating system login requirements on the target UNIX machine (the machine where the server is to run). However, the server must have **write** permission to the top-level Encina directory and its mirror directory. By default, Enconsole defines servers with the user ID **encina** and the group ID **encina**. If you are

using the Enconsole defaults, you must create these user and group identifiers on the target machine. Enconsole automatically grants the **encina** user ID the necessary operating system **write** permission to Encina directories and files.

- **For Servers to Be Run on Windows NT Systems**. There are no local operating system login requirements on the target Windows NT machine (the machine where the server is to run). Enconsole ignores the user ID and group ID fields on the Process Options form of the server definition form; you do not have to create the **encina** user and group identifiers on the target machine.

Perform the following steps to define a server:

1. Choose the **Define Server** command from the **Actions** menu, and then choose the type of server to define. After you choose a server type, Enconsole displays the server form for that type.
2. Type the name of your server in the **Name** field on the server form. For example, type **sfs1** for an SFS server named **sfs1**. By default, Enconsole creates the CDS pathname **/.:/***monitor_cell***/server/***server_name*, where *monitor_cell* is the name of your Monitor cell and *server_name* is the name of the server.
3. From the **Node** option list, select the name of the node on which the server is to run. For Encina recoverable servers, Enconsole displays the appropriate volume fields after you select a node.
4. If the server is an Encina recoverable server, specify the log volume disk name for the server.

   For Servers to Be Run on UNIX and Windows NT Systems. In the **Log Volume Disk Name** field, type the name of the disk on which you plan to store the transaction log data for your server. On UNIX, specify the name of a raw disk partition—for example, **/dev/rsd1g**.

   On Windows NT, a disk is an entire physical disk, a logical disk drive (partition), or an operating system file. To specify an entire physical disk, use the form **\\.\PHYSICALDRIVE***number_of_drive* where *number_of_drive* specifies the number (zero-based) of the physical disk (for example, **\\.\PHYSICALDRIVE3**). To specify a logical disk drive (partition), use the form **\\.***partition_letter* where *partition_letter* specifies the letter of the partition (for example, **\\.\D:**). To specify a file, use the full pathname of an operating system file, including the partition letter—for example, **D:\filevol1**.

   For Servers to Be Run on AIX Systems that use operating system logical volumes. In the **Log Volume Disk Name** field, type the name of the AIX logical volume on which you plan to store the transaction log data for your server.

5. If the server is an Encina recoverable server, specify the log volume mirror disk name for the server. Mirroring is not required, but it is strongly recommended.

   For Servers to Be Run on UNIX and Windows NT Systems. In the **Log Volume Mirror Disk Name** field, type the name of the disk on which you plan to store the mirror, or backup, of the server transaction log data.

   For Servers to Be Run on AIX Systems that use operating system logical volumes. Enconsole is not used to specify the log volume mirror disk name.

6. If the server is an Encina recoverable server that requires data volumes, specify the data volume disk name for the server.

   For Servers to Be Run on UNIX and Windows NT Systems. In the **Data Volume Disk Name** field, type the name of the disk on which you plan to store the data for your server.

   For Servers to Be Run on AIX Systems that use operating system logical volumes. In the **Data Volume Disk Name** field, type the name of the AIX logical volume on which you plan to store the data for your server.

7. If the server is an Encina recoverable server that requires data volumes, specify the data volume mirror disk name for the server. Mirroring is not required, but it is strongly recommended.

   For Servers to Be Run on UNIX and Windows NT Systems. In the **Data Volume Mirror Disk Name** field, type the name of the disk on which you plan to mirror the data for your server.

   For Servers to Be Run on AIX Systems that use operating system logical volumes. Enconsole is not used to specify the data volume mirror disk name.

8. Provide additional information that is specific to your server. The server definition forms for Monitor application servers, generic servers, DCE application servers, and Encina Toolkit servers require that you enter the full pathname of the executable file that starts the server and the command-line arguments for the server. Monitor application servers also require that you enter the names of the interfaces that are exported by the server. For information about each server type, see "Choosing a server type" on page 41.

9. Choose the **Process Options** button on the server form. Enconsole displays the Process Options form as shown in Figure 32 on page 62. Verify the default values on the Process Options form.

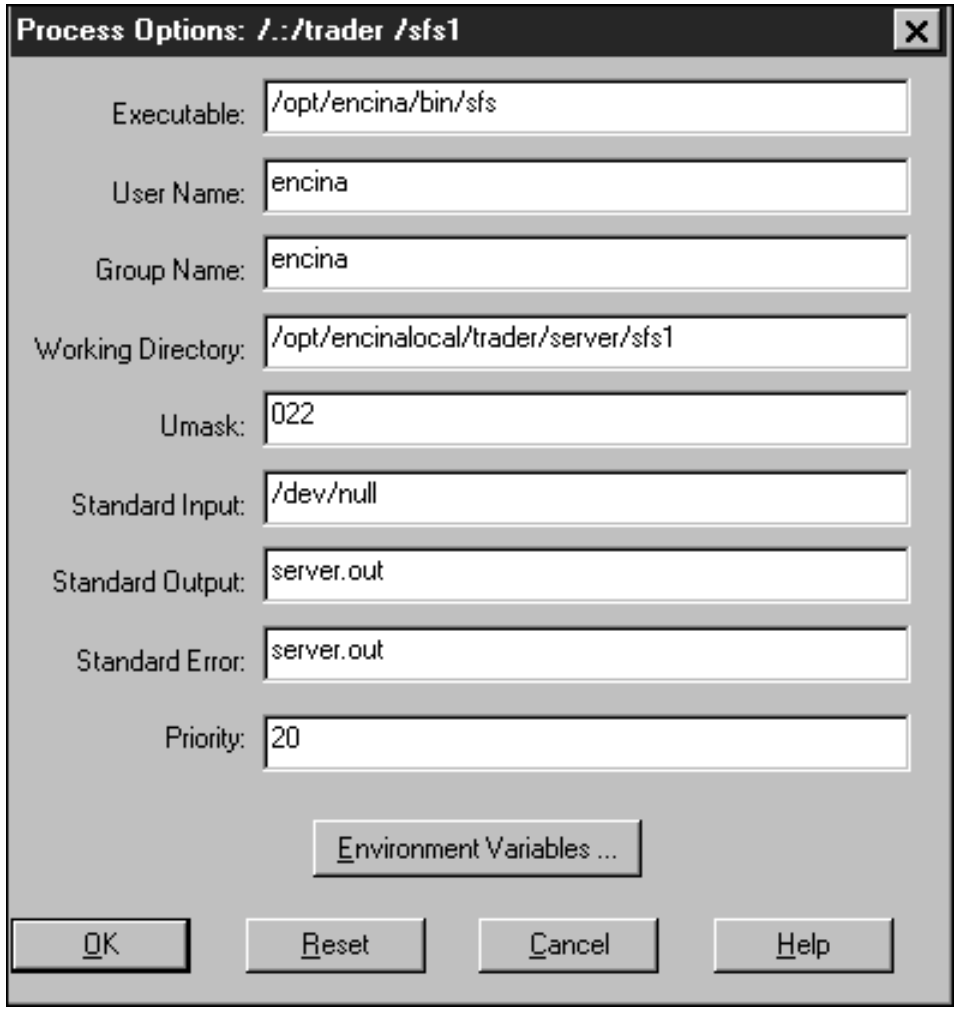

Figure 32. Process Options form for a server

For Servers to Be Run on UNIX Systems. The values of the User ID and Group ID fields must match the user and group IDs to be used on the target UNIX platform (the machine where the server is to run).

For Servers to Be Run on Windows NT Systems. Enconsole ignores the user ID and group ID fields of the Process Options form.

Choose the **OK** button to accept the default values for the remaining fields on the Process Options form and return to the server form.

10. When you have verified that your server definition is complete, choose the **OK** button on the server form. Enconsole stores the server configuration and displays the Command Status display screen.
11. Choose the **OK** button to close the Command Status display screen.

Enconsole updates the Encina Console display screen to show the new server's `Defined` state. You can now start the server as described in "Starting servers".

Starting servers

This section describes how to start a server that is defined or stopped. To start a server, you must be logged into DCE as the Encina administrative principal from within Enconsole.

- If you start a server with a `Defined` state (a server that was never started), the node manager creates files in the local file system and in the mirrored file system to store and organize the server output, restart, and keytab files. (By default, servers running in a Monitor cell store output files in the working directory *directory_root*/*monitor_cell*/**server/***server_name*, where *directory_root* is the name of the top-level Encina directory, *monitor_cell* is the name of the Monitor cell, and *server_name* is the name of the server. For example, the default working directory on most UNIX platforms and on Windows NT systems is **/opt/encinalocal/***monitor_cell*/**server/***server_name*.) Enconsole also performs the following DCE administrative tasks for Encina servers:
  - Creates a DCE principal for the server. The name of the principal is derived from the CDS entry for the server.
  - Adds the principal to the group **encina_servers_group**.
  - Creates a keytab file in the server's working directory on the machine on which the server runs.
  - Adds ACL entries to the server's ACLs in the CDS namespace to grant the necessary permissions to the appropriate groups and principals.

  It can take a few minutes for the system to start the server.
- If you start an Encina recoverable server (such as an SFS or RQS server) that is in a `Stopped` state, the server's log file is used to recover its volumes and enable it for full operation. The log file contains a description of modifications made to data volumes and is used to recover from a system or media failure. The restart files also play a role in the warm start of a server. These files are used to store volume configuration information.
- If you start a server that was cleaned as described in "Erasing servers" on page 64, Enconsole reallocates the necessary resources, such as the server's working directory and files, and the server is cold started.

Perform the following steps to start a server:

1. Choose the **Start Server** command from the **Actions** menu. Enconsole displays a selection box that lists the servers that can be started.
2. Select the server that you want to start, and then choose the **OK** button. Enconsole closes the selection box and displays the Command Status display screen.

   Note: If the **Start Server** command does not succeed, review the errors displayed on the Command Status display screen. You can review additional messages stored in the server's output file (**server.out**, by default) in the server's working directory on the machine where the server is running.
3. Choose the **OK** button to close the Command Status display screen.

Enconsole updates the Encina Console display screen to show the server's `Running` state.

Administering servers

This section describes how to stop and erase servers. To erase a server means to remove its output, restart, and log files and change its state to `Defined`. You must erase a server before moving, cold starting, or deleting it.

Stopping servers

This section describes how to stop servers. You must be logged into DCE as the Encina administrative principal to stop a server with Enconsole. Perform the following steps to stop a server:

1. Choose the **Stop Server** command from the **Actions** menu. Enconsole displays the selection box that lists the servers that can be stopped. Enconsole does not display servers that are already stopped.
2. Select the server that you want to stop, and then choose the **OK** button. Enconsole closes the selection box, displays the Command Status display screen, and stops the server.
3. Choose the **OK** button to close the Command Status display screen.

Erasing servers

To move, cold start, or delete a server that has already been started, you must first erase it. Erasing a server removes the server output, restart, and log files that were created when the server was started and returns the server's state to `Defined`. You must erase a server before you can perform the following tasks:

- Move the server to a different node—After you erase the server, modify the **Node** field on the server definition form. Use the Enconsole **Query/Modify Server** command to change the **Node** field and restart the server as described in "Starting servers" on page 63. Note that before moving the server, you must set up the new node on which the server is to run; see *Planning and Installation Guide*.

- Cold start the server—After you erase the server, Enconsole reallocates the necessary resources, such as the server's working directory and restart files, the next time you start the server.
- Delete the server— After you erase the server, you can delete it as described in "Deleting objects".

CAUTION:
Erasing a server deletes its data. Erase a server only when you no longer need its data.

Perform the following steps to erase a server:

1. Stop the server to be erased. See "Stopping servers" on page 64 for more information.
2. Choose the **Erase Server** command from the **Actions** menu. Enconsole displays a selection box that lists the servers that can be erased.
3. Select the server that you want to erase, and then choose the **OK** button. Enconsole closes the selection box and displays the following message, where *server_name* is the name of your server:

   ```
   Erasing server "server_name" will erase its
   restart data. The server will need to be cold started
   once the operation completes. Proceed?
   ```

4. Choose the **Yes** button. Enconsole closes the message box, displays the Command Status display screen, and executes the **Erase Server** command.
5. Choose the **OK** button to close the Command Status display screen.

The server's output, restart, and log files are removed. You can now cold start the server as described in "Starting servers" on page 63 or delete it as described in "Deleting objects".

Deleting objects

When you use the Enconsole **Delete** command to delete an object (a server, server set, or server schedule), the DCE Cell Directory Service (CDS) entry for the object is also deleted. Other DCE resources (such as directories, principals, and keytab files) are not deleted. You must use DCE administrative commands to delete these resources (see your DCE documentation). When you use the **Delete** command to delete a Monitor application server, the associated interfaces are deleted from the CDS namespace if the interfaces are not associated with any other Monitor application servers.

The following steps describe how to delete a server. The procedure is the same for deleting server sets and schedules.

Perform the following steps to delete a server:

1. Stop the server to be deleted. See “Stopping servers” on page 64.
2. Erase the server. See “Erasing servers” on page 64.
3. Choose the **Delete Server** command from the **Actions** menu. Enconsole displays a selection box that lists the servers that can be deleted.
4. Select the server that you want to delete, and then choose the **OK** button. Enconsole closes the selection box, displays the Command Status display screen, and deletes the server.
5. Choose the **OK** button to close the Command Status display screen.

Part 2. Server maintenance and recovery

Chapter 3. Creating volumes

This chapter presents basic concepts and procedures for managing data storage in Encina. It describes how to define and allocate Encina volumes, add and expand volumes, add and remove mirrors, and perform other volume management tasks.

Data storage in Encina

An important task in defining a server is allocating physical storage—specifying where to store application data (data created by users of the system) and where to store log data (transaction state information and descriptions of changes to data).

Toolkit servers use the Toolkit Volume Service to manage their physical storage. The Volume Service removes the physical boundaries of standard devices (disks) by providing a volume. A *volume* is an abstract representation of storage space that can contain storage areas from one or more disks. For instance, the **xyz** volume can have storage space allocated from both **disk1** and **disk2**. Thus, Toolkit servers can use arbitrarily large volumes to store application and log data.

Volumes can be used as *log volumes* (to store a server's log files) or as *data volumes* (to store a server's application data). Each Toolkit server requires one volume on which to store its log file and one or more volumes on which to store its application data. Whether they are used as log or data volumes, Encina volumes are managed in the same way.

Encina volumes can use the following physical devices:

- **On non-AIX UNIX**. Volumes can be stored on disks (disk partitions).
- **On AIX**. Volumes can be stored on raw disk partitions or on AIX logical volumes.
- **On Windows NT**. Volumes can be stored on a physical disk, a logical disk drive (partition), or one or more fully allocated operating system files. These files can reside on a single disk (logical drive) or span multiple disks.

Physical and logical volumes

The Volume Service uses two types of volumes: physical volumes and logical volumes. A *physical volume* is a collection of disk partitions used to store all server data. Physical volumes have a maximum size of 16 TB. Because a physical volume can contain any portion of one or more disks, you must specify several characteristics of a physical volume when creating it. Disk

space is divided into regions, regions are divided into chunks, and chunks are divided into pages. The common unit of measurement is a page. Figure 33 illustrates the concepts of disk divisions, including a disk, region, chunk, and page.

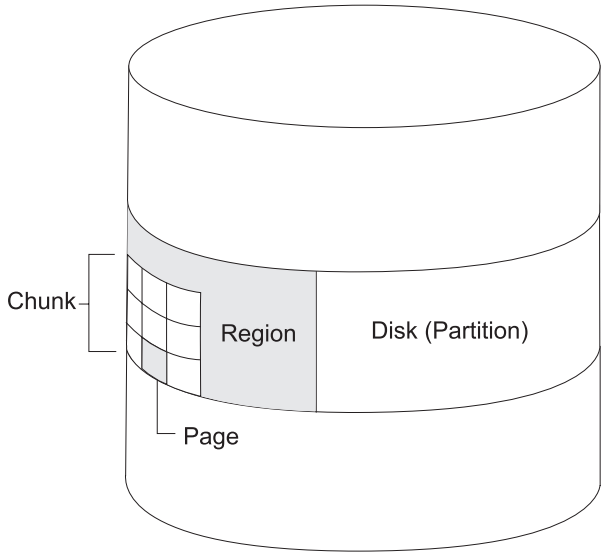

Figure 33. A disk and its units of allocation

A *disk* is defined as byte-addressable random access secondary storage. A disk is identified by a disk name, which is a system-specific printable character string. In most versions of UNIX, a disk is actually a disk partition; in this case, you specify the name of the disk partition—for example, **/dev/rsd2c**. On Windows NT, you can specify an entire physical disk, a logical disk drive (partition), or a fully allocated operating system file for the disk name.

Note: If you are using a physical disk, you must ensure that it does not contain any configured partitions. Encina cannot access a physical disk containing existing partitions.

A *region* is a collection of pages with contiguous addresses on a single disk. The size of a disk region should be a multiple of the chunk size. A region size that is not a multiple of the chunk size is rounded down to the nearest exact multiple; the rounding process results in wasted storage space. For example, if the chunk size is 32 pages, a region size specified as 72 pages is rounded down to 64 pages. A region cannot be smaller than a chunk.

A *chunk* is a unit of space allocation that can be used by Toolkit applications. Chunk size is the number of contiguous pages on a disk (provided that the disk supports that number of contiguous pages). Chunk size is important for

applications that require their data to be contiguously written to disk to achieve the highest possible I/O performance. It impacts how efficiently disk space is used and the amount of system overhead incurred when allocating space.

A chunk is larger than a page but smaller than a region. Chunk size is specified as a number of pages; the number must be a power of 2. The recommended chunk size is 64 pages. When you use Enconsole to create a physical volume for storing Toolkit application data, Enconsole automatically allocates a chunk size of 64. All physical volumes backing a logical volume must use the same chunk size. In most cases, it is best to accept the default chunk size. See "Chapter 9. Modifying server behavior" on page 183 for more information on chunk size.

A *page* is the smallest addressable unit of space administered by the Volume Service. Page size is a fixed Encina-configured parameter defined to be 4096 bytes; it is not a device specification.

Physical volumes are managed by logical volumes. A *logical volume* presents a user with a contiguous address space; that is, a logical volume simulates one large contiguous storage space by using regions of different disks. One logical volume can manage one physical volume and its mirrors (identical copies of the same data). A logical volume manages physical volumes by controlling all I/O and by mounting and dismounting the volumes. Note that a logical volume cannot be larger than the total of all physical disks that back the logical volume. Figure 34 on page 72 shows a logical volume that is composed of two physical volumes, each of which is a collection of regions of disks.

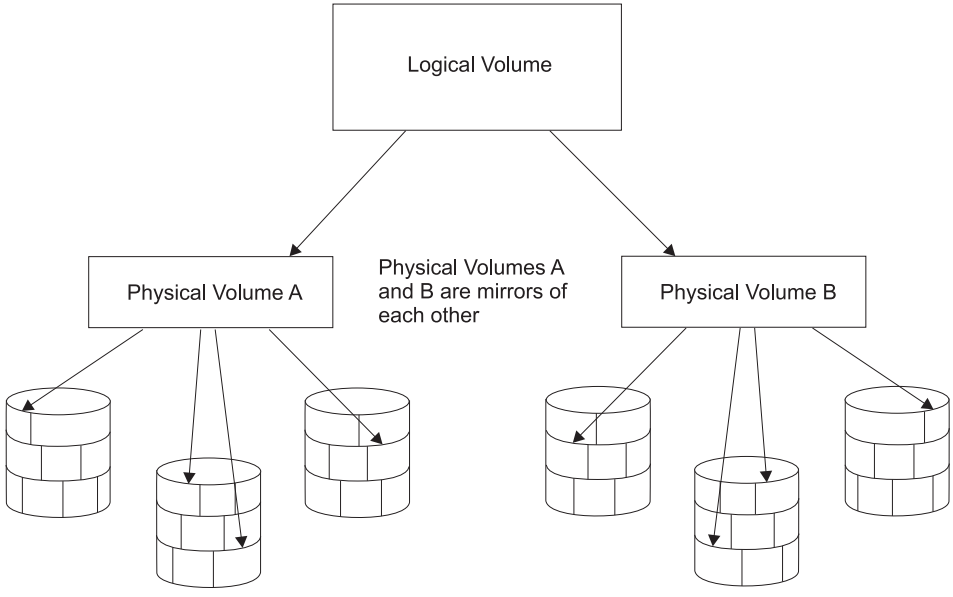

Figure 34. Physical and logical volumes

Creating physical devices for volumes

Before you can allocate a volume for Encina storage, you must create a physical device to be used as the volume. The type of physical device required depends on your platform. Encina supports the following storage devices:

- **On Non-AIX UNIX systems**. Encina supports raw disk partitions and UNIX file devices. Use a UNIX utility to create one or more disk partitions. Use an operating system utility such as **dd** to create file devices. Because disk I/O is faster than file I/O, it is strongly recommended that raw disk partitions be used in production environments and performance benchmarking.

  Note: To prevent buffering, use character devices instead of block devices. For easy replacement of a failed disk you can use symbolic links to the disk partitions.

- **On AIX systems**. Encina supports AIX logical volumes, raw disk partitions, and UNIX file devices. Note that you must use an AIX utility to create an AIX logical volume. In addition, you must use the AIX Logical Volume Manager (LVM) to manage AIX logical volumes. Use the Encina LVM to manage volumes stored on raw disk partitions and UNIX file devices on AIX.

- **On Windows NT systems**. Encina supports physical disks, logical disk drives (partitions), or one or more fully allocated operating system files. Use the Disk Administrator to create partitions. Use the Encina program **fileVol** (or your own program) to create file devices. Because disk I/O is faster than file I/O, it is strongly recommended that raw disk partitions be used in production environments and performance benchmarking.

  Note: If you are using a physical disk, make sure that it does not contain any configured partitions.

Using file devices as disks

You can use file devices as disks. File devices are created by using operating system utilities such as UNIX **dd**. File devices can be useful for development environments in which quick prototyping of systems does not require large production-size disks. However, it is strongly recommended that file devices not be used in production systems or for performance benchmark testing for the following reasons:

- File I/O can be buffered. Encina log data must be reliably written (not buffered).
- Performance is poor.

On UNIX systems. To create file devices, you can use the UNIX **dd** command. For example, the following command creates an 8-MB file device on UNIX:

```
% echo x | dd seek=16k of=/opt/vols/rqsDataVol
```

On Windows NT systems. If you are using files as volumes, you must create one or more fully allocated operating system files. You can use the Encina **fileVol** program (or your own program) to create the files. The **fileVol** program creates a fully allocated operating system file. The command syntax is

```
fileVol file_name file_size
```

Specify the name of the file to create as the *file_name* argument and the size of the file (in bytes or kilobytes) as the *file_size* argument. Specify bytes as an integer and kilobytes as an integer followed by the letter **k**. For example, the following command creates a fully allocated 4000- kilobyte operating system file named **D:\rqs1data**:

```
C:\> fileVol D:\rqs1data 4000k
```

The operating system files can be located on a disk partition (logical drive) or on a volume or stripe set that spans multiple disks. A file can be made to encompass an entire disk or disk partition. The use of network or floppy disk files for storing Encina logical volumes is strongly discouraged. Do not store the files in a compressed file system or on RAM disks. A Windows NT File System (NTFS) partition is the best choice.

The shaded portions in Figure 35 represent files. As shown, a volume can be a single file or multiple files on the same logical drive (a logical drive can be a Windows NT volume set). An operating system file can also encompass an entire physical disk or disk partition (logical drive).

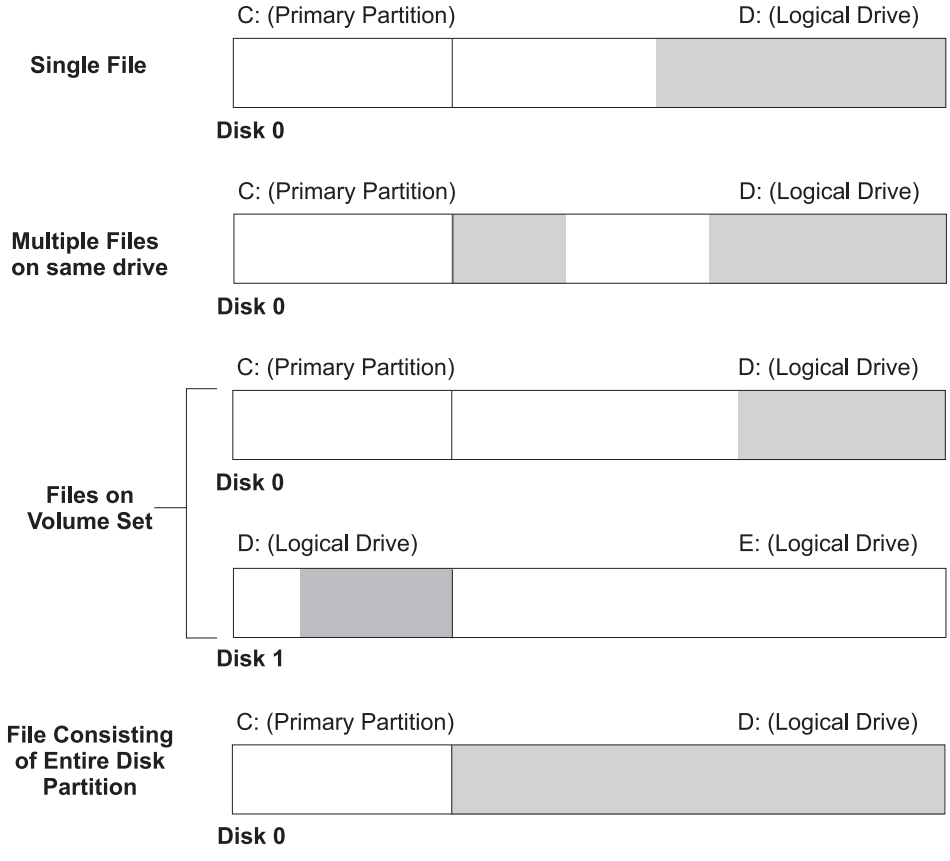

Figure 35. Files used as disks on Windows NT

The full pathname of the operating system file (including the drive letter) should be specified as the disk name for any *disk_name* arguments in Encina commands—for example, **D:\serverdata**. Files used as volumes have the same units of space allocation—regions, chunks, and pages—as disks have.

Allocating volumes

Enconsole configures a server (initializes the physical devices and creates volumes) when the server is started. This section describes how to allocate a simple volume configuration during the initial definition of a server. To allocate a more complex volume configuration or to allocate volumes after a

server is cold-started (or after you have defined a server but not yet started it), refer to the procedures in "Configuring complex volumes" on page 86.

Each server definition form contains fields for entering disk names for log volumes, data volumes, and mirrors. The forms differ depending on the type of server you are defining and the volumes required by the server.

- Encina servers (such as cell managers, node managers, SFS servers, RQS servers, and PPC Gateway servers) require one volume to store log data and (optionally) one mirror.
- Cell managers, SFS servers, and RQS servers also require at least one volume to store application data and (optionally) mirrors.

Note: Enconsole server definition forms allow you to specify only one disk per volume. You can allocate more complex volume configurations—for example, build multidisk volumes and add more mirrors—by using the Logical Volume Options form. Refer to "Configuring complex volumes" on page 86 for details.

Prerequisites for allocating volumes

To allocate volumes for cell managers and Encina recoverable servers, you need to provide the following:

- Names of the raw disk partitions or logical volumes (at least 8 megabytes each) to be used for the following:
  - Log volume
  - Data volume
- Optionally, names of the raw disk partitions or logical volumes (at least 8 megabytes each) to be used for the following:
  - Log volume mirror
  - Data volume mirror

To allocate volumes for node managers, you need to provide the following:

- Names of the raw disk partitions or logical volumes (at least 8 megabytes each) to be used for the log volume.
- Optionally, names of the raw disk partitions or logical volumes (at least 8 megabytes each) to be used for the log volume mirror.

The size of a mirror should be equal to the size of its primary volume (the original log or data volume it is mirroring). Note that mirrored logical volumes can be used as a backup. Mirrors are not required, but they are strongly recommended.

Note: If AIX logical volumes are used, mirroring is performed by the AIX Logical Volume Manager (LVM).

For detailed information about the software and hardware requirements for Encina, see *Planning and Installation Guide*.

When allocating volumes, keep the following information and cautions in mind:

- Encina volumes cannot be shared among Encina servers. For example, an Encina cell manager and Encina node manager cannot share the same log volume.
- Do not oversize Encina volumes. Unused space cannot be reclaimed.
- Only the server using the volume should access the disks. For example, you can give UNIX ownership to an Encina account (the identity under which the server runs) and give only the owner UNIX write permission to the disk. The minimum UNIX permissions needed to access a device is 600.

Specifying device names in Enconsole

To allocate a volume, supply Enconsole with the name of a physical device (for example, a raw disk partition or AIX logical volume) to be used as the volume. The way in which devices are specified to Enconsole depends on the platform, as follows:

- On UNIX, a disk is a raw disk partition or file device. To specify a raw disk partition name on Solaris, use the form **/dev/rdsk/***partition*—for example **/dev/rdsk/ct0d03**). To specify a raw disk partition on HP-UX, use the form **/dev/***volume_group***/r***logical_volume_name*—for example, **/dev/vg01/rrqsdata1**. (Encina sees HP-UX volumes as raw disk partitions.)
- On AIX, specify the name of the AIX logical volume—for example **/dev/lv01**. If you are using the Encina LVM on AIX, specify the raw disk partition name using the form **/dev/rdsk/***partition*—for example **/dev/rdsk/ct0d03**).
- On Windows NT, a disk is an entire physical disk, a logical disk drive (partition), or an operating system file. To specify an entire physical disk, use the form **\\.\PHYSICALDRIVE***number_of_drive* where *number_of_drive* specifies the number (zero-based) of the physical disk (for example, **\\.\PHYSICALDRIVE3**). To specify a logical disk drive (partition), use the form **\\.***partition_letter* where *partition_letter* specifies the letter of the partition (for example, **\\.\D:**). To specify a file, use the full pathname of an operating system file, including the partition letter—for example, **D:\filevol1**.

Allocating volumes (UNIX and Windows NT)

Perform the following steps to allocate volumes for UNIX and Windows NT servers.

Note: For AIX servers that use operating system logical volumes for Encina volumes, see "Allocating volumes (AIX logical volumes)" on page 78.

1. Create the physical devices to be used for your volumes. Refer to "Creating physical devices for volumes" on page 72.
2. Choose the **Define Server** command from the **Actions** menu, and then select the server from the Selection box. Enconsole displays the server definition form.
3. Specify a disk name for the log volume. In the **Log Volume Disk Name** field, type the name of the disk partition on which you plan to store the transaction log data for your server. Figure 36 shows the **Log Volume Disk Name** field on the RQS form.

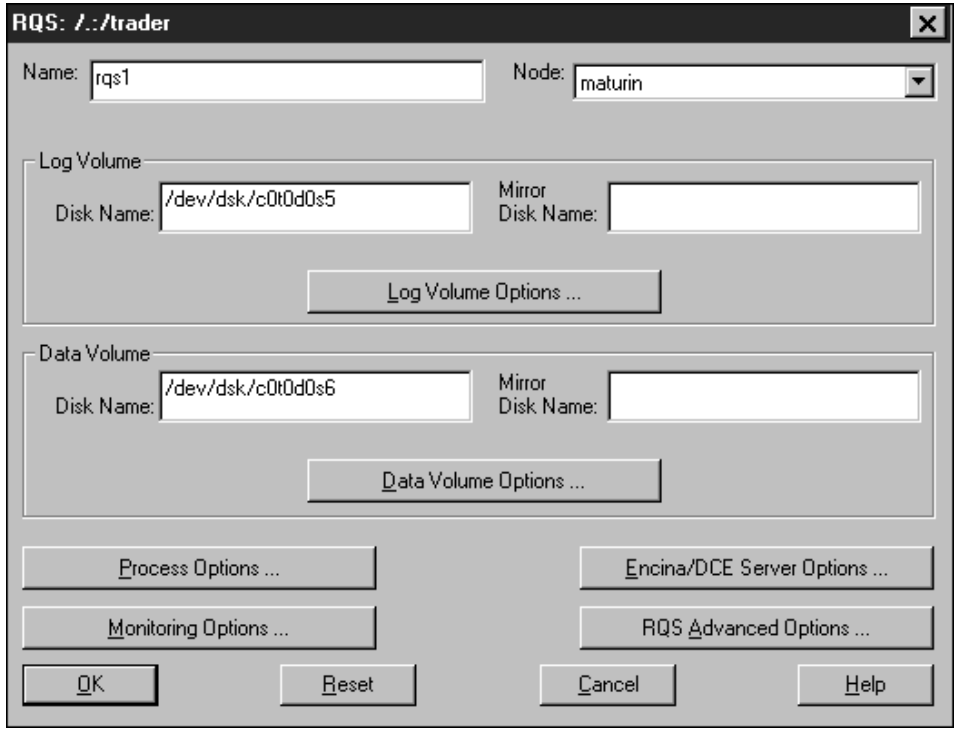

Figure 36. RQS form

4. (Optional). Specify a disk name for the log volume mirror. In the **Log Volume Mirror Disk Name** field, type the name of the disk partition on which you plan to store the mirror, or backup, of the transaction log data. Mirrors are not required, but they are strongly recommended.
5. Specify a disk name for the data volume. In the **Data Volume Disk Name** field, type the name of the disk partition on which you plan to store the data for your server. Figure 36 shows the **Data Volume Disk Name** field on the RQS form.

6. (Optional). Specify a disk name for the data volume mirror. In the **Data Volume Mirror Disk Name** field, type the name of the disk partition on which you plan to store the backup of the server data. Mirrors are not required, but they are strongly recommended.
7. Choose the **OK** button to confirm and save your entries.

Note: Volumes can be restored by using backups and log archive files. If you want a server to automatically generate archive files, use the **tkadmin enable mediaarchiving** command to enable media archiving. The server must be in the `Running` state. You must issue the command at an operating system prompt—you cannot use Enconsole to enable media archiving. Media archiving can be enabled for any server that manages data in Toolkit volumes (cell managers, SFS servers, and RQS servers) but not for a server that stores only log data (node managers and PPC gateway servers). Refer to "Chapter 6. Performing backups" on page 121 for details on media archiving.

Allocating volumes (AIX logical volumes)

Perform the following steps to allocate volumes for an AIX server that uses operating system logical volumes for Encina volumes:

1. Use an AIX utility to create the AIX logical volumes to be used for Encina volumes.
2. Choose the **Define Server** command from the **Actions** menu, and then select the server from the Selection box. Enconsole displays the server definition form.
3. Specify the AIX logical volume name for log data. In the **AIX Logical Volume Name** field, type the name of the AIX logical volume on which you plan to store the transaction log data for your server.

   Note: No data volumes are required for PPC gateway servers; therefore, a data volume field does not appear on the PPC Gateway Server form. Also, a mirror volume field does not appear; mirroring is handled by the AIX operating system.
4. Choose the **OK** button to confirm and save your entries.

Default Enconsole volume names

Enconsole uses the following default names for volumes:

- *For log volumes*, the default logical volume name is **logVol**. The default physical volume name is **logVol_physicalVol***number*.
- *For data volumes*, the default logical volume name is **dataVol**. The default physical volume name is **dataVol_physicalVol***number*.

You can view the default names of physical and logical volumes on the Logical Volume Options form for a data or log volume. You access this form

from the server definition form. Figure 37 shows an example Logical Volume Options form for a log volume.

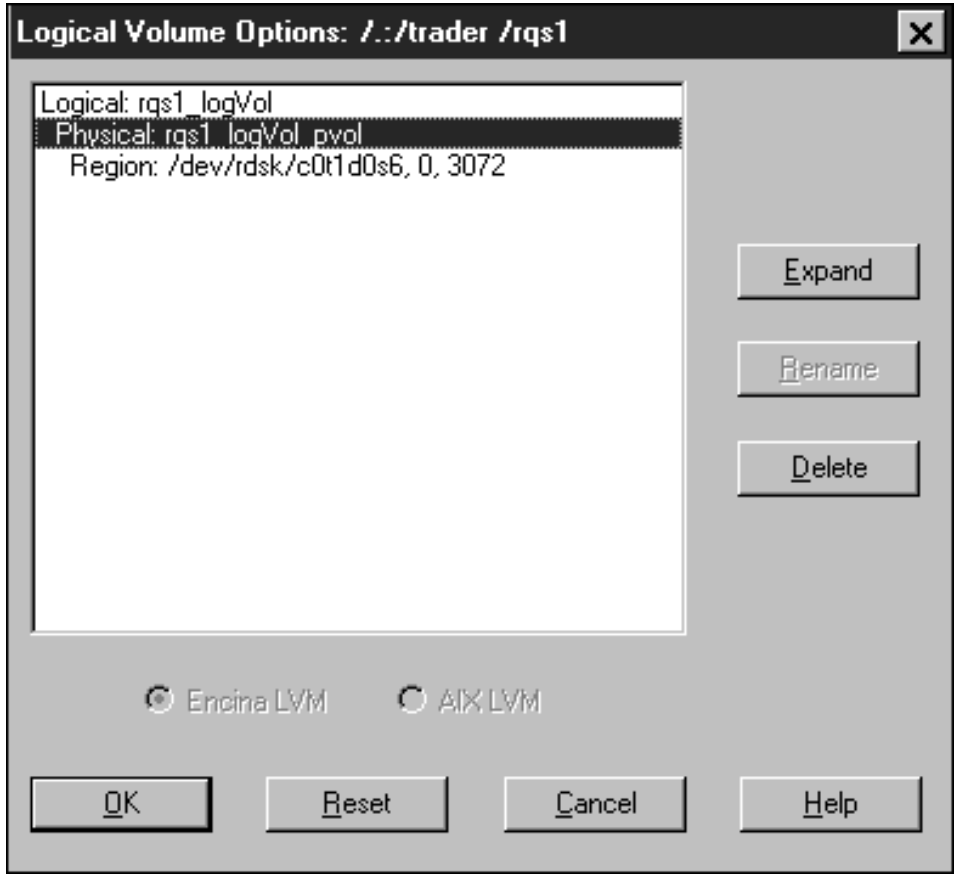

Figure 37. Example Logical Volume Options form

Note that the default volume names created by Enconsole are case-sensitive. If you use a default volume name as an argument to an Encina administrative command (such as the **tkadmin rename lvol** command), you must type the name exactly as shown (with uppercase and lowercase letters). You can use Enconsole to change the default names. See "Renaming and relocating volumes" on page 101 for details.

Displaying logical volume information

To view information about a logical volume, use either Enconsole or the Encina **tkadmin** commands. Enconsole provides information about a volume's structure (which physical volumes back a logical volume and which disk regions make up the physical volume). The **tkadmin** commands provide more

detailed information about configuration and usage of logical volumes. Each method of displaying information is described in the following sections.

Displaying information with Enconsole

The Enconsole Logical Volume Options form displays a hierarchical view of a logical volume, its backing physical volumes, and the disks making up the physical volumes. Figure 37 on page 79 shows an example Logical Volume Options form. The `Logical` line contains a logical volume name. The `Physical` line contains a physical volume name. The `Region` line contains the name of a disk partition, the offset used, and the number of pages used.

Displaying information with tkadmin commands

You can use the **tkadmin** commands to view volume information. The following **tkadmin** commands retrieve information about volume configuration and usage:

- To retrieve the names of disks, physical volumes, and logical volumes, use the following commands:
  - **tkadmin list lvols**
  - **tkadmin list pvols**
  - **tkadmin list disks**
- To retrieve detailed usage information about volumes, use the following commands:
  - **tkadmin query lvol** (for *any* logical volume)
  - **tkadmin query logvol** (for a logical volume used for logging)
  - **tkadmin query pvol** (for a physical volume)
  - **tkadmin query disk** (for a disk)

The **tkadmin** commands must be issued at an operating system prompt.

For example, the **tkadmin query pvol** command displays the total size (in kilobytes) of a physical volume, its chunk size, descriptions of its layout on one or more disks, and the name of the logical volume to which it is mapped.

On UNIX. In the following example, the physical volume **pvol1** is queried:

```
%  tkadmin query pvol pvol1

Information about physical volume pvol1
All sizes and offsets are in pages, Page Size is 4 Kbytes
Mapped to logical volume lvol1
Chunk Size: 8
numRegions: 2
region 0: disk: /dev/rsd1f size: 496
region 1: disk: /dev/rsd1e size: 496
total size: 992
```

On Windows NT. In the following example, the physical volume **rqsDataPVol** is queried:

```
C:\> tkadmin query pvol rqsDataPVol

Information about physical volume rqsDataPVol
All sizes and offsets are in pages. Page Size is: 4 Kbytes
Mapped to logical volume rqsDataVol
chunkSize: 64
numRegions: 1
region 0:  disk:  \\.\D:  offset:  0  size:  960
total size: 960
```

See the **tkadmin** reference pages for syntax and example output.

Mirroring logical volumes

A *mirror* (also called a *replica*) is a copy of data. You can mirror logical volumes to increase the availability of the data they store. Mirroring is strongly recommended for both log and data volumes. You should *always* mirror log volumes; log data is required to restore application data in the event of a media failure. You can create mirrors prior to starting a server (when you are initially allocating volumes for the server). You can also add mirrors at any time while a server is running. Enconsole automatically synchronizes the mirrors.

It is recommended that mirrors reside on different devices so that single-point disk failures cannot damage both copies of a volume. Figure 38 on page 82 illustrates an example configuration in which multiple physical disks are used to store log and server data on Windows NT. Mirrors for log and server data are placed on separate physical disks. This typically improves performance by allowing log and data reads or writes to occur in parallel and by localizing reads and writes to only one volume, reducing the time spent waiting to access different regions of a disk with more than a single volume.

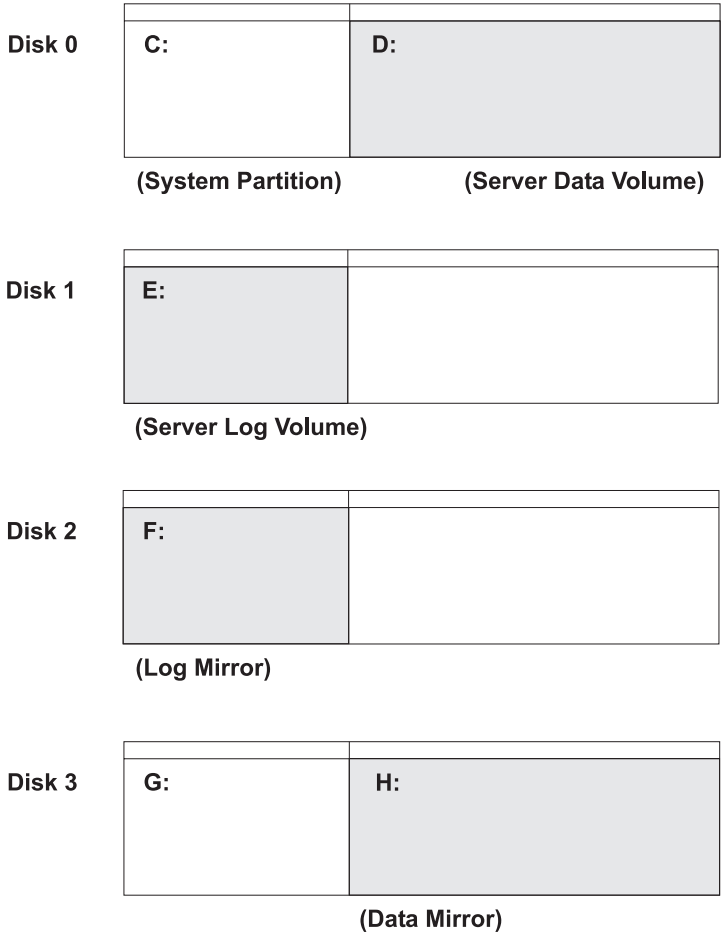

Figure 38. Example disk mirroring on Windows NT

Considerations for mirroring

You can mirror data by using either an operating system facility or Encina's Volume Service. On the AIX operating system using the AIX LVM, you can mirror an AIX logical volume by using an AIX utility (not Encina). The Windows NT operating system provides mirroring by using partitions in mirrored sets. Also, some machines are equipped with special hardware that transparently replicates data. Before choosing NT mirrored sets or hardware-based replication in place of Encina replication, you should understand the recoverability issues described in the following text.

There are several factors to consider when selecting mirroring policies and mechanisms. First, recall the following typical properties of disks:

- They are permanent, except in the event of a failure. Mirroring, backups, and logging are meant to protect against failures. In the case of Encina, logging is also involved in providing transactional guarantees and can be used to bring backups up to date so that nothing is lost between the time of the last backup and the time of a failure.
- They store very large amounts of data and therefore rely on secondary storage (typically disk-shaped magnetic media).
- They are slow (compared to primary storage). In particular, they often must wait for an internal component (the head) and the media they use to physically move so that the head is over that very small portion of the media involved in each individual data transfer. This typically involves moving the head radially in or out and then waiting for rotation to bring the proper portion of the spinning media under the head. This is a high-latency operation and reduces the potential throughput for the entire disk.
- They store data in pages (4 KB). Each page includes extra information to ensure that it either contains consistent data or contains no usable data. The page is thus an atomic unit of data storage. An entire page is either successfully stored with the extra information or it is not and is reported as bad when a read is attempted.

Mirroring introduces additional complexities. When data is mirrored, writing to all copies of the data at once is not desirable. Simultaneous writes to disk greatly increase the likelihood of multiple copies becoming bad should a failure occur while the data is being written.

Besides the risk of bad copies, mirroring introduces the possibility of unmatched copies. Comparing each copy for every read is expensive. The typical way to deal with this problem is to find (and correct) any inconsistencies when a system is brought back up after a failure. When the amount of data is large and the importance of bringing the system up quickly is high, a simple scheme such as scanning each page breaks down. Encina implements a sophisticated logging scheme that addresses this and other issues.

Another potential problem can occur when a write that needs to be atomic spans multiple pages, and only some of those pages get written to a copy of the data. This is addressed by Encina and may not be by other mirroring schemes.

These considerations generally apply to both operating-system and hardware-based data mirroring for two or more copies of the data. In summary, Encina mirroring is likely to be the most reliable approach. The performance difference between Encina and hardware-based schemes is potentially significant. However, before choosing a hardware-based scheme, be

sure that it provides the required reliability. The performance difference between Encina and operating system schemes may favor Encina in cases where both the operating system and Encina are logging data; if the operating system is not logging data, the reliability of its mirroring is reduced but the performance may be slightly higher.

This document assumes that you are using Encina's Volume Service for mirroring on UNIX and Windows NT systems. It does not apply to AIX operating system logical volumes managed with the AIX Logical Volume Manager.

Adding and removing mirrors

You can add a mirror to a volume when the server is being defined or when it is running. The procedure for adding a mirror is similar to the procedure for expanding (adding) a volume.

Adding a mirror

Perform the following steps to add a mirror to a volume. This example adds a mirror to a running server.

1. Choose the **Query/Modify** command from the **Actions** menu, and then select the server from the Selection box. Enconsole displays the server definition form.
2. Determine which volume to mirror—the log volume or the data volume. Choose either the **Log Volume Options** or **Data Volume Options** button. Enconsole displays the Logical Volume Options form for the selected volume.
3. Select the **Logical:** *name_of_logical_volume* line on the form, as shown in Figure 39 on page 85.

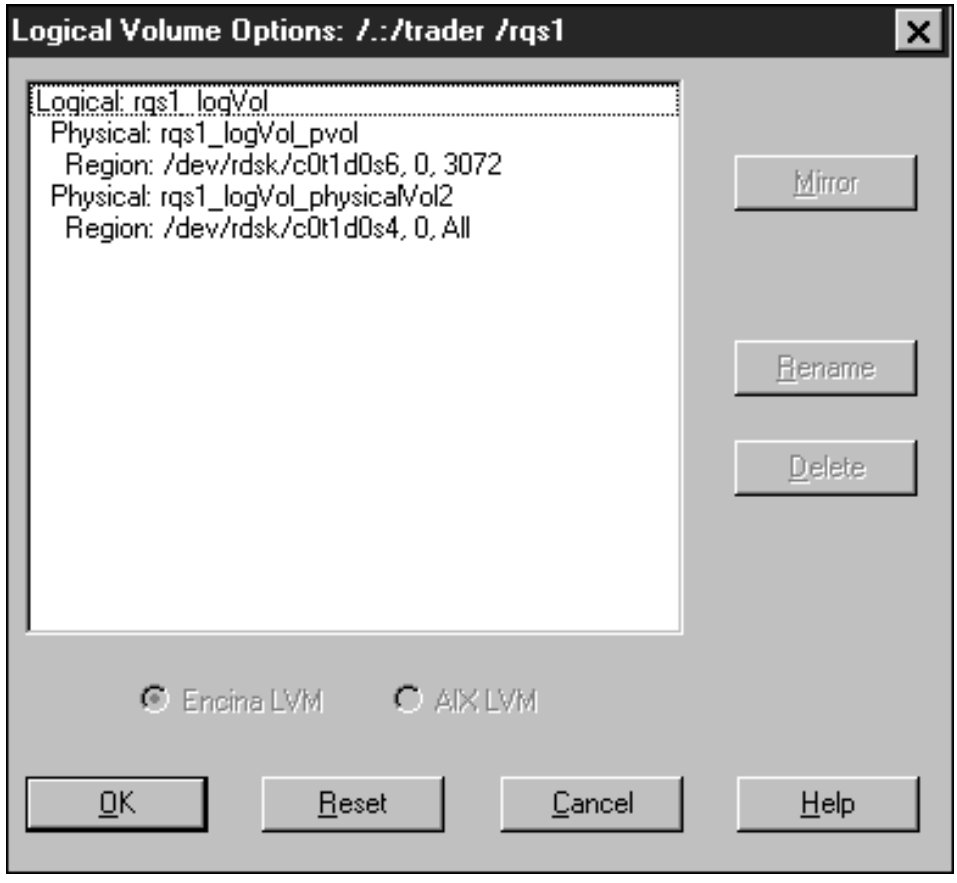

Figure 39. Volumes and mirrors displayed on Logical Volume Options form

4. Choose the **MIRROR** button. Enconsole displays the Disk Region form. Type a disk name in the **Disk Name** field and modify the values of the other fields if necessary. Choose the **OK** button to confirm your entries and to close the Disk Region form.
5. View the updated information on the Logical Volume Options form and choose the **OK** button to confirm the changes and close the form. Be sure that the size of the new physical volume equals the size of the existing physical volume or volumes.
6. Choose the **OK** button on the server definition form. (Enconsole does not make the changes until you confirm the changes on the server definition form.) Enconsole then initializes the new disk, creates a physical volume, and adds and synchronizes the mirror.

Note: Enconsole server definition forms have limited space. The forms provide fields for specifying just two disks per volume: one for the

original volume and one for its mirror. If your original volume is backed by multiple disks, or if you have multiple mirrors, the server definition form cannot display all the disk names. Instead, all volume information is displayed on the Logical Volume Options form. (The disk fields no longer appear on the server definition form.)

Removing a mirror

Perform the following steps to remove a mirror. This example removes a mirror from a running server.

1. Choose the **Query/Modify** command from the **Actions** menu and then select a server from the Selection box. Enconsole displays the server definition form.
2. Determine which volume to remove the mirror from—the log volume or the data volume. Choose either the **Log Volume Options** or **Data Volume Options** button. Enconsole displays the Logical Volume Options form for the selected volume.
3. Select the **Physical:** *name_of_physical_volume* line corresponding to the mirror to be deleted.
4. Choose the **DELETE** button.
5. View the updated information on the Logical Volume Options form and choose the **OK** button to confirm the changes and close the form.
6. Choose the **OK** button on the server definition form to confirm the changes. Enconsole removes the mirror.

If you want to reclaim space occupied by a deleted mirror, refer to the procedures in "Reclaiming storage space (UNIX and Windows NT)" on page 105 or "Reclaiming storage space (AIX logical volumes)" on page 107.

Configuring complex volumes

"Allocating volumes" on page 74 describes how to use Enconsole to allocate volumes for a server. By default, Enconsole supplies (on the server definition form) fields for specifying only one disk per log volume, one disk per data volume, and one disk per mirror. This is probably sufficient for prototype systems. However, production systems can require more volumes. Once the original volume and its mirror have been specified on the server definition form, you perform additional volume configuration tasks by using the Logical Volume Options form. This form is accessed from the **Log Volume Options** or **Data Volume Options** button on the server definition form. This chapter describes how to use the Logical Volume Options form to configure additional volumes and mirrors.

In Enconsole, use the Logical Volume Options form to do any of the following:

- Expand a logical volume (that is, add more disk space to the physical volumes that back it). (See “Expanding volumes” on page 88.)
- Build multidisk physical volumes.
- Add logical volumes to SFS servers. (SFS servers can have multiple logical volumes for storing application data.)
- Add and remove mirrors. (See “Adding and removing mirrors” on page 84.)

The Logical Volume Options form displays a hierarchical view of a logical volume and its associated physical volumes and regions. Figure 40 shows a volume configuration on UNIX.

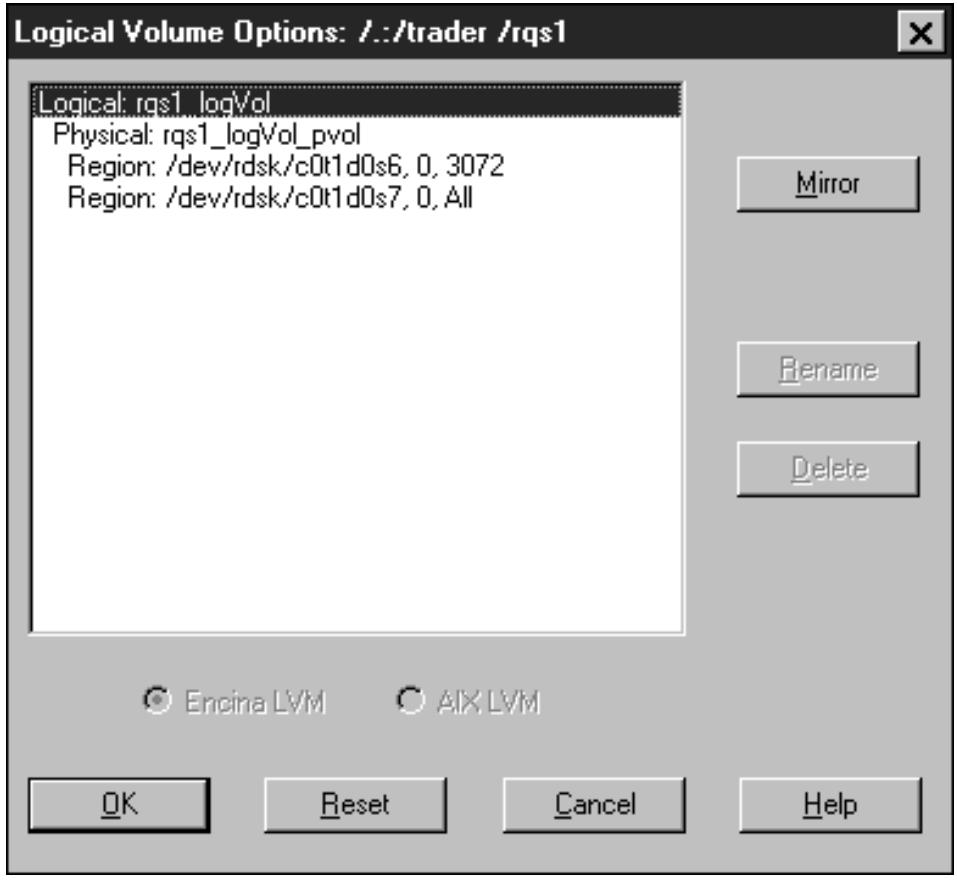

Figure 40. Physical volume using two disks

You can make changes to a logical volume at the following times:

- During the initial definition of a server
- When a server’s state is `Defined` or `Running`

The procedures used to expand physical volumes, build multidisk physical volumes, add SFS logical volumes, and add mirrors are similar. The following steps summarize the general procedure.

1. Access the server definition form. If you are initially defining a server, choose the **Define Server** command from the **Actions** menu, and then select a server from the Selection box. If you are making changes to a defined (but not yet started) server or to a running server, choose the **Query/Modify** command from the **Actions** menu, and then select a server from the Selection box.
2. Access the Logical Volume Options form (choose either the **Data Volume Options** button or the **Log Volume Options** button, depending on which volume you are modifying).
3. Select a line on the Logical Volume Options form. To expand a physical volume or build multidisk physical volumes, select the **Physical:** *name_of_physical_volume* line. To add a mirror, select the **Logical:** *name_of_logical_volume* line. To add an SFS logical volume, select the **Logical:** *name_of_logical_volume* line.
4. Choose one of the following:
   - The **EXPAND** button (for expanding a volume or building a multidisk physical volume)
   - The **MIRROR** button (for adding a mirror)
   - The **ADD** button (for adding an SFS logical volume)

   Enconsole displays the Region Disk Form.
5. Specify the new volume information on the Region Disk Form.
6. Verify your choices by choosing the **OK** button on the Logical Volume Options form and then the **OK** button on the server definition form.

If you make changes to a volume as part of the initial server definition or when a server is defined but not yet started, the changes take place as soon as the server is started. If you make changes to a volume while a server is running, the changes take place immediately.

Expanding volumes

When a server requires additional storage space for its data, you can expand the server's logical volume. Expanding a logical volume means that you are adding additional disk space to the physical volume (or volumes) that back it. Expansion appends new empty pages to the end of a logical volume, as shown in Figure 41 on page 89.

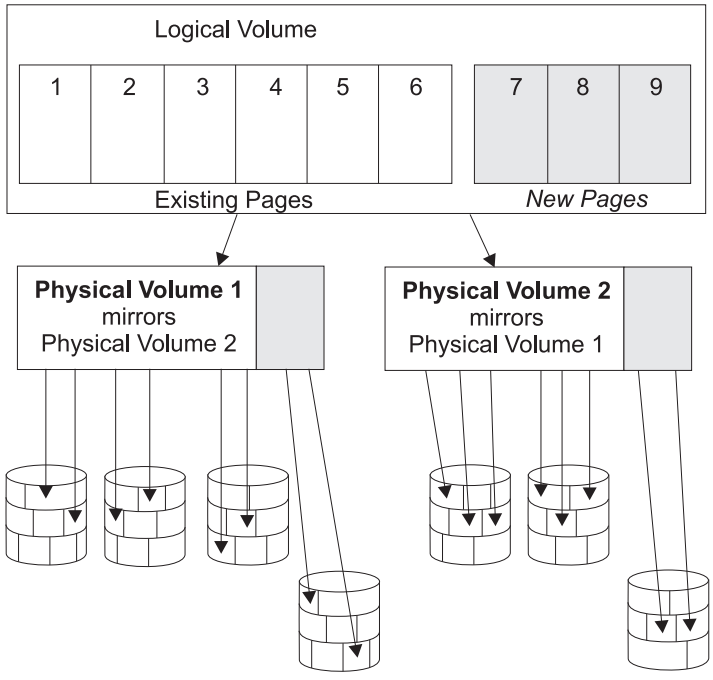

Figure 41. Expanding Encina volumes

The procedures for expanding volumes and for adding volumes (building multidisk volumes) are the same—in both cases you are adding disk space to a physical volume. When you expand a logical volume (add additional disk space to its physical volumes) remember to expand all physical volumes backing the logical volume, including all mirrors.

Note: If AIX operating system logical volumes are being used, first use an AIX utility to expand the AIX logical volume. Then use Enconsole to expand the Encina logical volume.

The following example creates a log volume that is backed by one disk, **/dev/rdsk/c0t1d0s6**. A second disk, **/dev/rdsk/c0t1d0s7**, is added during the initial definition of the server. Perform the following steps to add disk space to a volume:

1. Choose the **Define Server** command from the **Actions** menu, and then select the server from the Selection box. Enconsole displays the server form. In the **Log Volume Disk Name** field, type the name of the disk that is to back the log volume. Figure 42 on page 90 shows the disk name **/dev/rdsk/c0t1d0s6**.

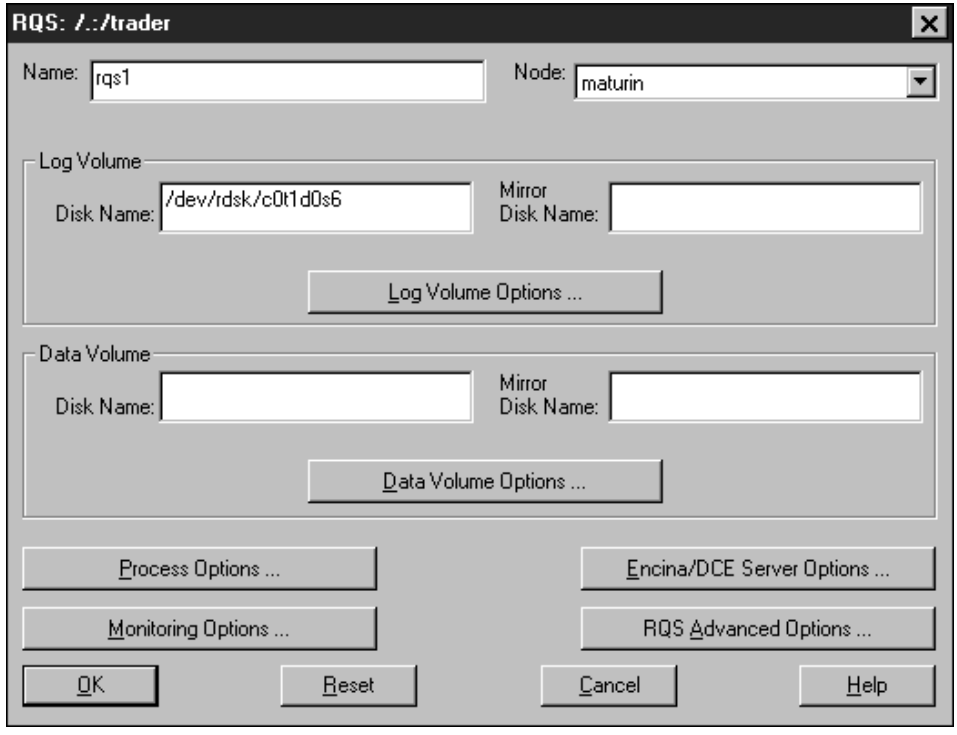

Figure 42. Log Volume Disk Name field

2. Choose the **Log Volume Options** button. Enconsole displays the Logical Volume Options form. Select the line for the physical volume to be expanded (for example, the `Physical: rqs1LogVol_pvol` line as shown in Figure 43 on page 91).

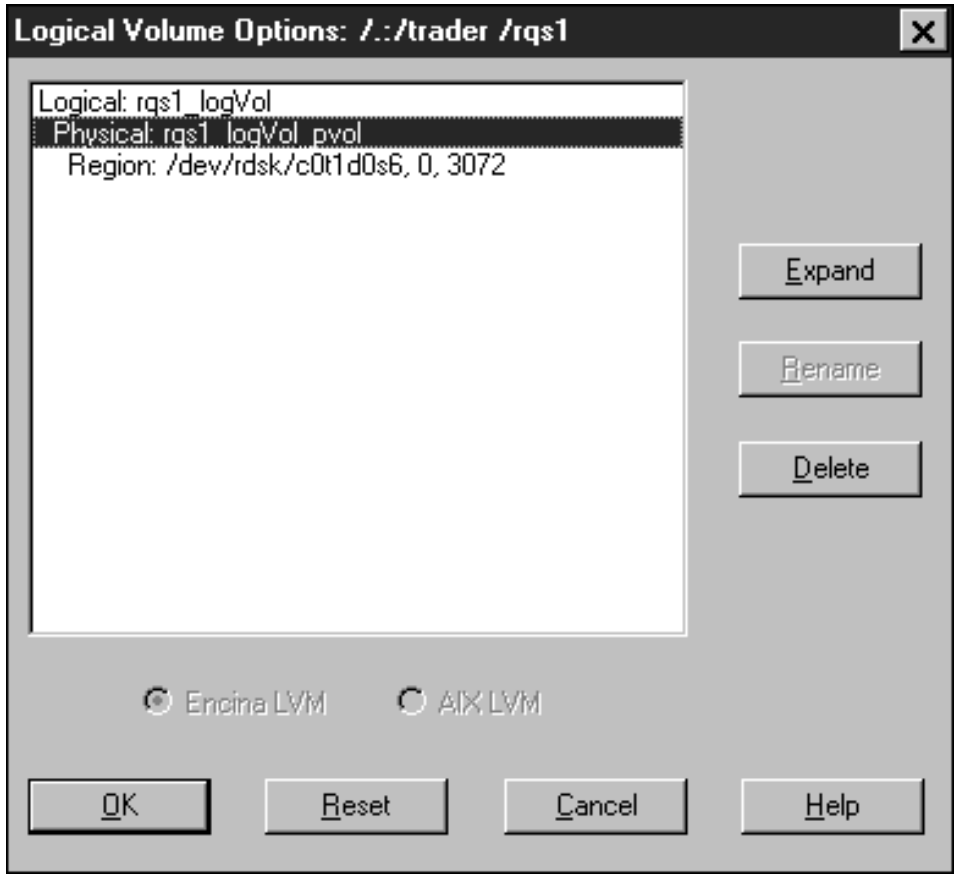

Figure 43. Physical volume selected

3. Choose the **EXPAND** button. Enconsole displays the Disk Region form. For each disk that you want to add to the volume, specify the following:
   - Disk name (path to the device) in the **Disk Name** field.
   - Disk offset in the **Offset** field. The default value is 0 (zero).
   - Number of pages in the **Num Pages** field. The default value is the entire disk partition (represented by the value `All`). A page is 4096 bytes (an Encina constant).

   Figure 44 on page 92 shows the example disk name **/dev/rdsk/c0t1d0s7**, the offset **0** (zero), and the default number of pages **All**.

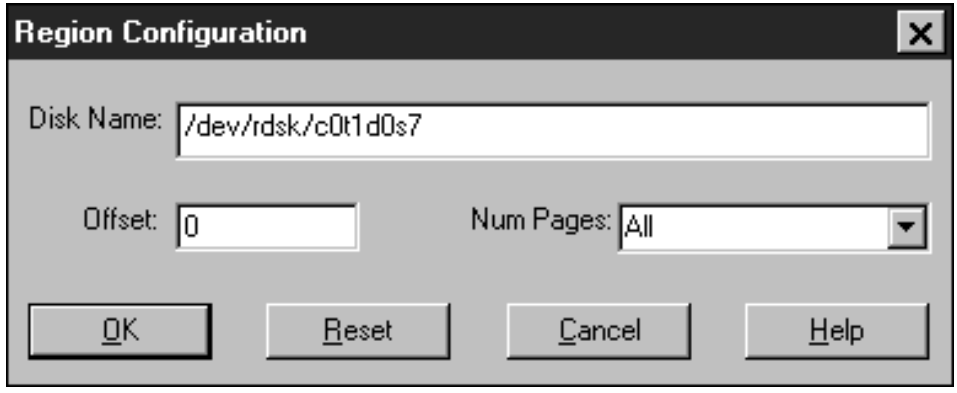

Figure 44. Disk Region form

4. Choose the **OK** button to confirm your entries and to close the Disk Region form. Figure 45 on page 93 shows that disks **/dev/rdsk/c0t1d0s6** and **/dev/rdsk/c0t1d0s7** both back the physical volume.

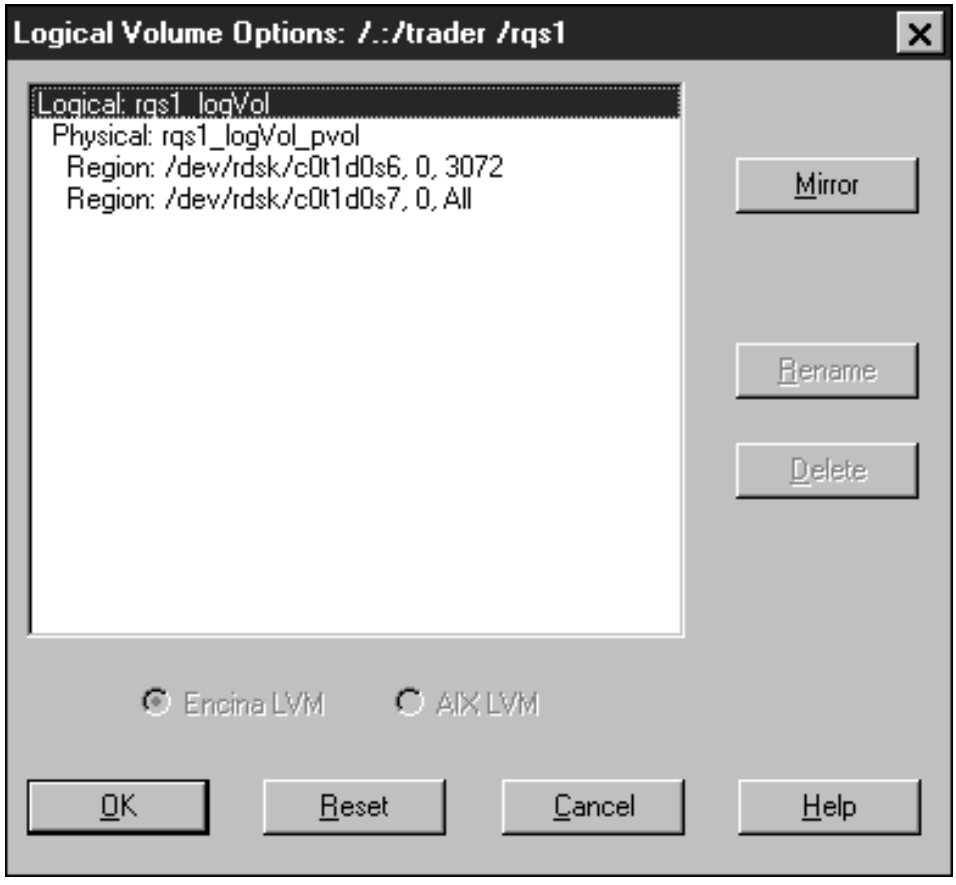

Figure 45. Physical volume using two disks

5. Choose the **OK** button to verify your entries and to close the Logical Volume Options form. Choose the **OK** button on the server definition form.

Note: Enconsole server definition forms have limited space. The forms provide fields for specifying just two disks per volume: one for the original volume and one for its mirror. If your original volume is backed by multiple disks, or if you have multiple mirrors, the server definition form cannot display all the disk names. Instead, all volume information is displayed on the Logical Volume Options form. (The disk fields no longer appear on the server definition form.)

Using part of a disk partition for a volume

You can use part of a disk partition for a volume. By default, when you supply a disk name, Enconsole allocates the entire disk partition. You must use Enconsole to modify the disk description and specify which portion of the disk is to be used.

The following example shows how to use part of a disk for the log volume of an SFS server on a UNIX system. The example allocates disk pages 0 (zero) through 14,999 of the disk **/dev/rdsk/c0t1d0s0** for the log volume. Perform the following steps to specify part of a disk partition for a volume:

1. Choose the **Define Server** command from the **Actions** menu, and then select the server from the Selection box. Enconsole displays the server form.
2. After specifying the server name and node, choose the **Log Volume Options** button. Enconsole displays the Logical Volume Options form. Select the line for the physical volume to be expanded (for example, the `Physical: rqs1LogVol_pvol` line on the form).
3. Choose the **EXPAND** button to specify the region description. Enconsole displays the Disk Region form. Type the disk name in the **Disk Name** field and the number of pages to be used in the **Num Pages** field. The default value of the **Num Pages** field is `All`, the entire partition. Figure 46 shows the example disk **/dev/rdsk/c0t1d0s0**, the default offset **0** (zero), and the example number of pages **15000**.

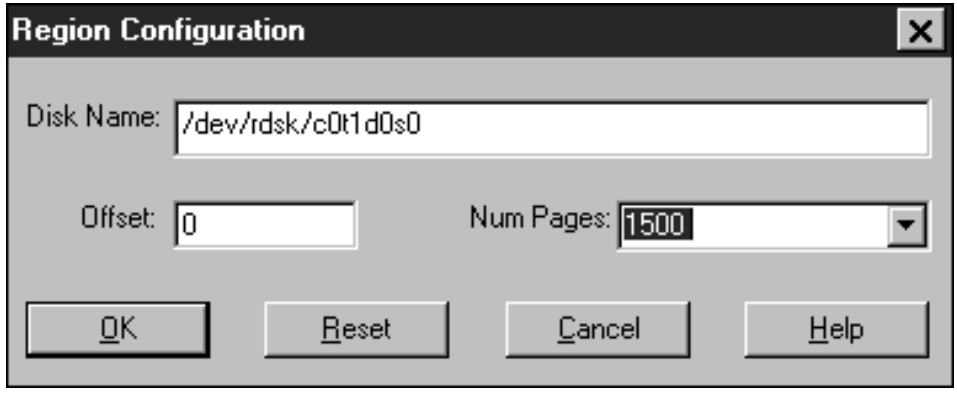

Figure 46. Log Volume Disk Region form

4. Choose the **OK** button to confirm your entries and to close the Disk Region form.
5. Choose the **OK** button to verify your entries and to close the Logical Volume Options form. Choose the **OK** button on the server definition form.

Chapter 4. Maintaining volumes

This chapter describes how to use administrative mode, how to rename and relocate volumes, and how to reclaim space from volumes no longer needed by a server.

Using administrative mode

This section describes when to use administrative mode and how to restart a server in administrative mode.

When to use administrative mode

Some administrative tasks require a server's volumes to be temporarily disabled. Restarting an Encina server in *administrative mode* temporarily disables its volumes. When restarted in administrative mode, a server

- Exports only its Toolkit administration remote procedure call (RPC) interface; it does not export any of its other RPC interfaces. For example, a Structured File Server (SFS) server restarted in administrative mode does not export the SFS client interface but exports only the interface needed by the **tkadmin** program.
- Does not perform recovery on its volumes (as it would if started in *normal operations mode* with enabled volumes).

The following tasks can be done only when a server is in administrative mode:

- Recovering from media failure (log or data volumes have been damaged and you need to restore the volumes). Note that administrative mode is not necessary to recover from damage to mirrors.
- Renaming logical and physical volumes (the **tkadmin rename lvol** and **tkadmin rename pvol** commands).
- Relocating physical volumes (the **tkadmin move pvol** command).
- Reclaiming storage space occupied by a logical volume (the **tkadmin delete lvol**, **tkadmin delete pvol**, and **tkadmin delete disk** commands).
- Performing any other operation that requires a volume to be offline.

The commands for renaming and relocating volumes are covered in "Renaming and relocating volumes" on page 101. The procedure for reclaiming storage space is covered in "Reclaiming storage space (UNIX and Windows NT)" on page 105 and "Reclaiming storage space (AIX logical volumes)" on page 107. The procedure for restoring data and log volumes is covered in "Chapter 7. Recovering from failures" on page 133.

To perform administrative-mode tasks, you must first restart a server in administrative mode and then use the appropriate **tkadmin** commands to administer the server. The following sections describe how to enter administrative mode by using Enconsole and by using the command line.

Restarting a server in administrative mode (using Enconsole)

A server must be restarted in administrative mode for operations such as moving, renaming, and restoring volumes. Perform the following steps to restart a server in administrative mode:

Note: See "Restarting a server in administrative mode (using the command line)" on page 98 for how to start a cell manager in administrative mode.

1. Stop the server. Choose the **Stop Server** command from the **Actions** menu, select the server from the Selection box, and then choose the **OK** button. Enconsole stops the server.
2. Choose the **Query/Modify** command from the **Actions** menu, select the server from the Selection box, and then choose the **OK** button. Enconsole displays the server form.
3. Access the Recovery Options form. Choose the **Encina/DCE Server Options** button from the server form. When Enconsole displays the Encina/DCE Server Options form, choose the **Recovery Options** button. Enconsole displays the Recovery Options form (shown in Figure 47 on page 97). Note that the **Recovery Options** button is available for Encina recoverable servers only.

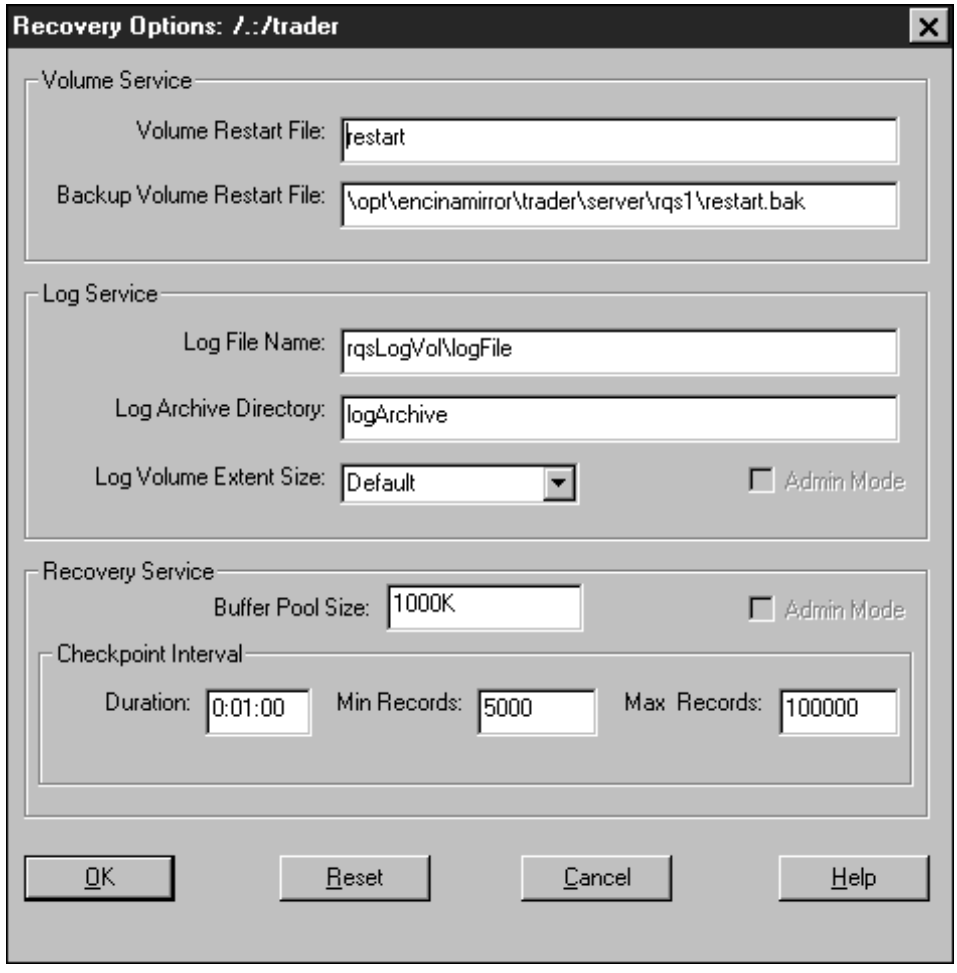

Figure 47. The Recovery Options form

4. Based on which volume you are administering, choose the appropriate check box for one of the following modes:
   - **Log Service Admin Mode** — Disables the log volume and data volume (disabling a log volume automatically disables a data volume). Use this mode if you are restoring log volumes.
   - **Recovery Service Admin Mode** — Disables the data volume only. Use this mode if you are restoring data volumes.
5. Verify and commit your choices. Choose the **OK** button from the Recovery Options form, choose the **OK** button from the Encina/DCE Server Options form, and choose the **OK** button from the server form. The Encina Console shows that the server state is `Stopped (Admin)`. This indicates that the server is stopped in administrative mode.

6. Start the server. Choose the **Start Server** command from the **Actions** menu, choose the server from the Selection box, and then choose the **OK** button. Enconsole starts the server, and the Encina Console shows that the server state is `Running (Admin)`. This indicates that the server is running in administrative mode.
7. At the operating system prompt, issue the appropriate **tkadmin** commands to administer your server. Enconsole does not provide administrative capabilities for a server in administrative mode. You must use **tkadmin** commands.
8. When you have finished using administrative commands, stop the server and turn off administrative mode. Choose the **Stop Server** command from the Enconsole **Actions** menu to stop the server, and then use the **Query/Modify** command to access the Recovery Options form. To turn off administrative mode, clear the appropriate check box on the Recovery Options form.
9. Choose the **OK** button from the Recovery Options form, choose the **OK** button from the Encina/DCE Server Options form, and choose the **OK** button from the server form. The Encina Console shows that the server state is `Stopped`. This indicates that the server is stopped and is no longer in administrative mode.
10. Start the server. Choose the **Start Server** command from the **Actions** menu, choose the server from the Selection box, and then choose the **OK** button. Enconsole starts the server, and the Encina Console shows that the server state is `Running`.

Restarting a server in administrative mode (using the command line)

You can start a server in administrative mode from the command line. To do this, use one of the following:

- The **enccp** program
- The administrative-mode option of a restart script (available for cell managers on UNIX systems only)
- A server-specific startup command (such as the **ecm**, **enm**, **rqs**, or **sfs** commands)

To use **enccp** to start a server in administrative mode, perform the following steps:

1. Start the **enccp** program and connect to your Monitor cell. For example, the following command starts **enccp** and connects to the Monitor cell **/.:/trader**:

```
% enccp -cell /.:/trader
enccp>
```

2. Issue the **enccp** server start command with either the **-recAdminMode** or **-logAdminMode** option. (The **-recAdminMode** option disables data volumes; the **-logAdminMode** option disables both log and data volumes.) The syntax is as follows:

   object_type **start** *object_name* [**-recAdminMode**] [**-logAdminMode**]

   For example, the following command starts the server **rqs1** in administrative mode and disables its data volumes:

   ```
   enccp> rqs start rqs1 -recAdminMode
   ```

3. Perform the required volume maintenance or recovery tasks by using **tkadmin** commands.
4. Issue the **enccp** server stop command. The syntax is as follows:

   object_type **stop** *object_name*

   For example, the following command stops the server **rqs1**:

   ```
   enccp> rqs stop rqs1
   ```

5. Issue the **enccp** server start command to restart the server.

To use a restart script to start a cell manager in administrative mode, perform the following steps:

Note: This procedure is for UNIX systems only.

1. Run the **rc.encina.cell** restart script and specify either the **-recAdminMode** or the **-logAdminMode** option. (The **-recAdminMode** option disables data volumes; the **-logAdminMode** option disables both log and data volumes.) The syntax is as follows:

   rc.encina.cell [**start**] [**-recAdminMode**] [**-logAdminMode**]

   For example, the following command starts a cell manager in administrative mode:

   ```
   % rc.encina.cell -recAdminMode
   ```

2. Perform the required volume maintenance or recovery tasks by using **tkadmin** commands.
3. Run the restart script and specify the **stop** argument. The partial syntax is as follows:

   rc.encina.cell stop

   For example, the following command stops a cell manager:

   ```
   % rc.encina.cell stop
   ```

4. Run the restart script to restart the cell manager in normal operations mode (do not specify any options with the restart script).

To use server-specific startup commands to start a server in administrative mode, perform the following steps:

1. Determine the original command-line options specified when the server was cold started.
2. Issue the startup command (for example, the **rqs** or **sfs** command). Use the original command-line options and add the following options:
   - **-a** Indicates *administrative mode*. Disables server volumes and starts the server in administrative mode.
   - **-A** *principal* Specifies a principal (called the *exclusive authority*), for example, **encina_admin**. When the server starts, this principal has sole access to administer the server and to create or modify initial access control lists (ACLs).
   - **-l** *log_information*. Specifies the name of the log file created at initial start. The log filename takes the form *log_volume*/*log_file_name*, where *log_volume* is the logical volume on which the file resides and *log_file_name* is the log filename. Specify the **-l** option if you are restoring data volumes. Specifying this option is equivalent to specifying the **-recAdminMode** option of **enccp** commands. Omit the **-l** option if you are restoring log volumes or doing other operations on log volumes. Omitting the option is equivalent to specifying the **-logAdminMode** option of **enccp** commands.

   For example, the following command starts a Recoverable Queueing Service (RQS) server named **rqs1** in administrative mode. Since logical volume names are specified (the **-l** option), only data volumes are disabled.

   ```
   %  rqs -a -A encina_admin -n /.:/branch1/server/rqs1 -l rqsLogVol/rqsLogfile
   -v "restart:/opt/encinamirror/branch1/server/rqs1/restart.bak"
   -p branch1/server/rqs1 -k keyfile -Z 1 -N 2
   ```
3. Perform the required volume maintenance or recovery tasks by using **tkadmin** commands.
4. Enable the server by using the **tkadmin enable server** command.

Table 5 on page 101 provides a summary of the steps necessary for starting a server in administrative mode, repairing volumes, and returning the server to normal operation. This summary applies to the use of server-specific startup commands (the **enm**, **ecm**, **rqs**, **sfs**, and **ppcgwy** commands only). See "Chapter 7. Recovering from failures" on page 133 for details on recovering volumes.

Table 5. Restarting a server in administrative mode

Restarting a Server in Administrative Mode	Restoring or Relocating Data Volumes	Restoring or Relocating Log Volumes
Restart the server	Issue the startup command with the **-a**, **-A**, **-n**, **-v**, and **-l** options, at minimum	Issue the startup command with the **-a**, **-A**, **-n**, and **-v** options, at minimum (*Omit log information)*
Restore (or relocate) volumes as necessary	**tkadmin restore lvols** and/or **tkadmin move pvol**	**tkadmin restore logvol** and/or **tkadmin move pvol**
Enable the log file (if log volumes are being restored)	(Not applicable)	**tkadmin enable logfile**
Restore the data volume (if log volumes are being restored)	(Not applicable)	**tkadmin restore lvols**
Recover the volumes	**tkadmin recover lvols**	**tkadmin recover lvols**
Enable the server	**tkadmin enable server**	**tkadmin enable server**

Renaming and relocating volumes

This section contains procedures for renaming and relocating volumes for organizational purposes. You can rename logical and physical volumes in Enconsole in the following situations:

- When you first allocate volumes (during the initial definition of a server, before it is cold started)
- When you add or expand volumes and have not yet confirmed your volume options (you have not yet selected the **OK** buttons on the Logical Volume Options and server definition forms)

Use the **Rename** button on the Logical Volume Options form to rename a volume in Enconsole.

After a server's volumes have been configured, however, you must use both Enconsole and **tkadmin** commands to rename volumes. The following procedures describe how to use Enconsole and **tkadmin** commands to rename volumes after they have been configured.

Renaming logical volumes (UNIX and Windows NT)

Perform the following steps to rename a logical volume:

Note: If AIX operating system logical volumes are used, see "Renaming volumes (AIX logical volumes)" on page 102 for instructions.

1. Restart the server in administrative mode. See “Using administrative mode” on page 95 for instructions.
2. At an operating system prompt, use the **tkadmin rename lvol** command to rename the volume. The command syntax is

   tkadmin rename lvol -server *server_name old_name new_name*

   You must specify the old name of the logical volume for the *old_name* argument and the new name of the logical volume for the *new_name* argument. For example, enter the following command to rename a logical volume named **sfsvol3** to **encinavol1**:

   ```
   % tkadmin rename lvol sfsvol3 encinavol1
   ```
3. Restart the server in normal operations mode.

Renaming physical volumes (UNIX and Windows NT)

Perform the following steps to rename a physical volume:

1. Restart the server in administrative mode. See “Using administrative mode” on page 95 for instructions.
2. Use the **tkadmin rename pvol** command to rename the volume. The command syntax is

   tkadmin rename pvol -server *server_name old_name new_name*

   You must specify the old name of the physical volume for the *old_name* argument and the new name of the physical volume for the *new_name* argument. For example, enter the following command to rename a physical volume named **sfspvol1** to **encinapvol1**:

   ```
   % tkadmin rename pvol sfspvol1 encinapvol1
   ```
3. Restart the server in normal operations mode.

Renaming volumes (AIX logical volumes)

The link between an Encina logical volume and an AIX logical volume is the name of the supplied AIX logical volume. Changing the name of an AIX logical volume that backs an Encina logical volume disrupts the physical storage of the server using that logical volume. If it is necessary to rename an AIX logical volume, the Encina logical volume must be unmapped and then remapped to the renamed AIX logical volume. If possible, avoid renaming AIX logical volumes mapped to Encina volumes.

Perform the following steps to remap an Encina logical volume to a renamed AIX logical volume:

1. Restart the server in administrative mode. See “Using administrative mode” on page 95 for instructions.
2. Use an AIX utility to rename the AIX logical volume.
3. Use the **tkadmin unmap lvol** command to unmap the Encina logical volume. The command syntax is

tkadmin unmap lvol -server *server_name logical_volume_name*

You must specify the name of the Encina logical volume for the *logical_volume_name* argument. For example, enter the following command to unmap a logical volume named **sfsvol3**:

```
% tkadmin unmap lvol sfsvol3
```

4. Use the **tkadmin remap lvol** command to remap the Encina logical volume. The command syntax is

   tkadmin remap lvol -server *server_name logical_volume_name operating_system_logical_volume_name chunk_size*

   You must specify the name of the Encina logical volume for the *logical_volume_name* argument, the new name of the AIX logical volume for the *operating_system_logical_volume_name* argument, and the chunk size of the Encina logical volume for the *chunk_size* argument. For example, enter the following command to remap a logical volume named **sfsvol3** to the renamed AIX volume **aixvol3**:

   ```
   % tkadmin remap lvol sfsvol3 aixvol3 64
   ```

   Note: Use the **tkadmin query pvol** command to determine the chunk size of a physical volume. If you used Enconsole to create the physical volume, its chunk size is 64. (The default chunk size used by Enconsole is 64.)

5. Restart the server in normal operations mode.

Relocating physical volumes

To manage disk usage, you can relocate a server's logical volume to a different collection of disks. A logical volume is relocated by moving its underlying physical volumes. To find the names of underlying physical volumes, view the appropriate Logical Volume Options form (for either the log volume or the data volume) in Enconsole or review the output of the **tkadmin query lvol** command.

Note: If you need to relocate physical volumes after a media failure, refer to the restore procedures in "Chapter 7. Recovering from failures" on page 133.

Perform the following steps to relocate a physical volume:

1. Restart the server in administrative mode. See "Using administrative mode" on page 95 for instructions.
2. If necessary, create a new physical disk (the destination disk). On UNIX, use an operating system utility to create a disk or disk partition. On Windows NT, use the Disk Administrator to create a new disk or the Encina **fileVol** program to create a new fully allocated operating system file.

Note: The disk space created for the destination disk must be at least as large as the disk space of the original (source) disk.

3. Use the **tkadmin init disk** command to initialize the disk. The syntax is

   ```
   tkadmin init disk -server server_name disk_name
   ```

   You must specify the name of the disk to be initialized for the *disk_name* argument.

   On UNIX. For example, enter the following command to initialize the disk **/dev/rsd1f**:

   ```
   % tkadmin init disk /dev/rsd1f
   ```

   On Windows NT. For example, enter the following command to initialize the disk **D:\serverdata**:

   ```
   C:\> tkadmin init disk D:\serverdata
   ```

4. Use the **tkadmin move pvol** command to move the physical volume. The syntax is

   ```
   tkadmin move pvol -server server_name physical_volume_name
   number_of_regions {{source_disk_name source_disk_offset
   region_size destination_disk_name destination_disk_offset}...}
   ```

   You must specify the name of the physical volume for the *physical_volume_name* argument and the number of regions for the *number_of_regions* argument. For each region, you must specify the source disk name for the *source_disk_name* argument, the source disk offset for the *source_disk_offset* argument, the size of the region for the *region_size* argument, the destination disk name for the *destination_disk_name* argument, and the destination disk offset for the *destination_disk_offset* argument. Disk offsets and the region size are measured in units of pages (4096 bytes).

   On UNIX. For example, enter the following command to move one region of a physical volume named **sfspvol2** from a disk named **/dev/rsd1f** (with an offset of 0 and a region size of 1984 pages) to a disk named **/dev/rsd2g** with an offset of 0:

   ```
   % tkadmin move pvol sfspvol2 1 /dev/rsd1f 0 1984 /dev/rsd2g 0
   ```

   On Windows NT. For example, enter the following command to move a physical volume named **sfspvol2** from a disk named **D:\serverdata** (with an offset of 0 and a region size of 2880 pages) to a disk named **F:\serverdata** with an offset of 0:

   ```
   C:\> tkadmin move pvol sfspvol2 1 D:\serverdata 0 2880 F:\serverdata 0
   ```

5. Restart the server in normal operations mode.

Reclaiming storage space (UNIX and Windows NT)

You can reclaim space occupied by a server's logical volumes by deleting the logical volume and its associated physical volumes and disks. (Be sure that a logical volume is no longer in use before reclaiming its storage space.) The underlying physical volumes and disks need not be deleted if they can be used by the same server to store other data. However, they must be deleted if you plan to use the space for another server.

You must delete logical volumes, physical volumes, and disks in that order. The following list summarizes the steps needed to reclaim storage space:

1. Remove all mirrors of the volume.
2. Restart the server in administrative mode.
3. Delete the logical volume.
4. Delete the physical volume or volumes.
5. Delete the disk or disks (if no other physical volume is stored on the disk).
6. (*Windows NT Only*) Delete the operating system files used as disks.

Note: If AIX operating system logical volumes are used, see "Reclaiming storage space (AIX logical volumes)" on page 107 for instructions.

The examples in this procedure reclaim the space occupied by the logical volume **sfslvol1**. Assume that the logical volume **sfslvol1** is backed by two physical volumes—the original physical volume, **sfspvol1**, and a mirror, **sfspvol1.mirr**. Each physical volume resides on its own UNIX disk partition—for example, **sfspvol1** on **/dev/rsd0f** and **sfspvol1.mirr** on **/dev/rse0f**.

Perform the following steps to reclaim storage space:

1. Remove all mirrors of the backing physical volume. You can use either Enconsole or the **tkadmin remove mirror** command. To use Enconsole, see the instructions in "Adding and removing mirrors" on page 84. The syntax of the **tkadmin remove mirror** command is

   tkadmin remove mirror -server *server_name logical_volume_name physical_volume_name*

   You must specify the name of the logical volume for the *logical_volume_name* argument and the name of the physical volume for the *physical_volume_name* argument. (You can determine which physical volume and mirrors back a logical volume by viewing the appropriate Logical Volume Options form for your server in Enconsole or by using the **tkadmin query lvol** command.)

   For example, enter the following command to remove a mirror named **sfspvol1.mirr** from a logical volume named **sfslvol1**:

```
% tkadmin remove mirror sfslvol1 sfspvol1.mirr
```

2. Restart the server in administrative mode. See "Using administrative mode" on page 95 for instructions.
3. Use the **tkadmin delete lvol** command to delete the logical volume. Deleting the logical volume removes the mapping of a physical volume to the logical volume it backs. You must delete a logical volume before you delete its backing physical volumes. The command syntax is

   tkadmin delete lvol -server *server_name logical_volume_name*

   You must specify the name of the logical volume for the *logical_volume_name* argument. For example, enter the following command to delete a logical volume named **sfslvol1**:

   ```
   % tkadmin delete lvol sfslvol1
   ```
4. Use the **tkadmin delete pvol** command to delete the physical volume. You must delete a physical volume before you delete (uninitialize) the disks that back it. The command syntax is

   tkadmin delete pvol -server *server_name physical_volume_name*

   You must specify the name of the physical volume to be deleted as the *physical_volume_name* argument. Be sure to delete all physical volumes, including physical volumes that back mirrors. For example, enter the following command to delete a physical volume named **sfspvol1**:

   ```
   % tkadmin delete pvol sfspvol1
   ```
5. Use the **tkadmin delete disk** command to delete each disk that backs a physical volume. Deleting a disk marks the disk as uninitialized. The command syntax is

   tkadmin delete disk -server *server_name disk_name*

   You must specify the name of the disk as the *disk_name* argument. Any attempt to delete a disk that still stores a physical volume fails. Remember that some physical volumes are stored on multiple disks and that a single disk can hold multiple physical volumes or portions of physical volumes.

   In this example, each physical volume resides on its own disk partition—for example, **sfspvol1** on **/dev/rsd0f** and **sfspvol1.mirr** on **/dev/rse0f**. You must delete both disk partitions.

   On UNIX. For example, enter the following command to delete a disk named **/dev/rsd0f**:

   ```
   % tkadmin delete disk /dev/rsd0f
   ```

   On Windows NT. For example, enter the following command to delete a disk named **D:\rqs1data**:

   ```
   C:\> tkadmin delete disk D:\rqs1data
   ```

6. (*Windows NT Only*) Delete the operating system file serving as a disk (the file initially created by using the **fileVol** program or your own program). Use any standard file deletion mechanism to delete the file.

Note: A server retains a logical volume's name even after the volume is deleted. Further, all volumes of a Toolkit server must have unique names. If you want to reuse the name of a deleted logical volume, you must rename it (change the stored name to a different name). Then the original volume name can be reused (the server no longer retains information about this volume name).

Reclaiming storage space (AIX logical volumes)

You can reclaim space occupied by a server's logical volumes. (Be sure that a logical volume is no longer in use before reclaiming its storage space.) On AIX, you reclaim space by unmapping the Encina logical volume and then deleting the backing AIX operating system logical volume. Perform the following steps to reclaim space occupied by an AIX logical volume:

1. Restart the server in administrative mode. See "Using administrative mode" on page 95 for instructions.
2. Use the **tkadmin unmap lvol** command to unmap the Encina logical volume from its backing AIX logical volume. The command syntax is

   ```
   tkadmin unmap lvol -server server_name logical_volume_name
   ```

   You must specify the name of the logical volume for the *logical_volume_name* argument. For example, enter the following command to unmap a logical volume named **sfslvol1**:

   ```
   % tkadmin unmap lvol sfslvol1
   ```
3. Use an AIX utility to delete the AIX logical volume. You can then reuse the space previously occupied by the volume.

Chapter 5. Managing server transactions

To effectively administer transactions, you need to understand the states of transactions, how transactions are identified across servers, and how transactions use locking. This chapter describes transaction concepts and terminology useful in reviewing and (if necessary) intervening in problematic unresolved transactions. It describes how to display transaction information in Enconsole and how to evaluate and act on transactions.

Encina transaction processing

A *transaction* is a set of operations that must be executed as a single unit and are used to move data between consistent states. A distributed transaction runs in multiple processes, potentially on many machines. Transactions exhibit the following properties (called the ACID properties):

- *Atomicity*. A transaction is either successful or unsuccessful. Either all of the operations that make up a transaction take effect or none take effect. A successful transaction is said to commit. An unsuccessful transaction is said to abort. Any operations performed by an aborted transaction are undone.
- *Consistency*. A transaction transforms distributed data from one consistent (correct) state to another.
- *Isolation*. The transactions appear to be sequential, acting as though one completed before the other began, even though they may run concurrently. The effects of a given transaction are not visible to other transactions until that transaction commits.
- *Durability* (also known as permanence). Once a transaction commits, its effects are permanent. A subsequent failure (such as abnormal program termination, communications failure, or hardware crash) does not alter the transaction's effects.

When a transaction commits, all actions associated with that transaction are written to a log. In the event of system problems, those actions are repeated if necessary when the system's recovery mechanism replays the log. When a transaction aborts, any changes made by the transaction are undone. Once a transaction is undone (rolled back), no evidence that the transaction was ever attempted remains outside of records in the transaction processing system's log.

The Two-phase commit protocol

In order to synchronize related pieces of work taking place in different processes, distributed transactions use a *two-phase commit protocol*. The protocol guarantees that the work is successfully completed by all the processes or not

performed at all. For example, modifications to data either all fail or all succeed. The goal is to ensure that each participant in a transaction takes the same action (commits or aborts).

During the lifetime of a transaction, any participant in the transaction can unilaterally decide to abort the transaction. Once all of the work of the transaction is complete, the application attempts to commit the work by invoking the two-phase commit protocol. In the first phase, the *prepare phase*, each participant is asked to prepare. Once a participant is prepared, it agrees to accept whatever outcome the coordinator decides on (it can no longer unilaterally abort the transaction).

The final outcome of a transaction (how the transaction ends) depends upon the individual outcome of each participant in the transaction. In the *resolution phase*, a transaction commits if all participants can commit or aborts if any one participant aborts.

The Encina two-phase commit protocol works as follows:

- One of the participants in the transaction becomes the coordinator of the transaction. The coordinator tracks the status of and communicates with other participants in the transaction.
- Each participant is given an opportunity to prepare its work. Each participant notifies the coordinator that it is ready to commit or to abort. A participant can abort its work at any time before the prepare phase; however, once the participant "votes" to commit, it cannot abort.
- The coordinator tallies the results from all participants.
- The coordinator informs each participant of the result of the vote. The coordinator instructs each participant to commit if all participants have voted to commit, or to abort if any one participant has aborted.
- Each participant takes the appropriate action.

As shown in Figure 48 on page 111, the prepare phase consists of all actions up until the time each participant notifies the coordinator that it is ready to commit (or to abort). The resolution phase begins when the coordinator instructs each participant to either commit or abort, and ends when all participants have completed their work.

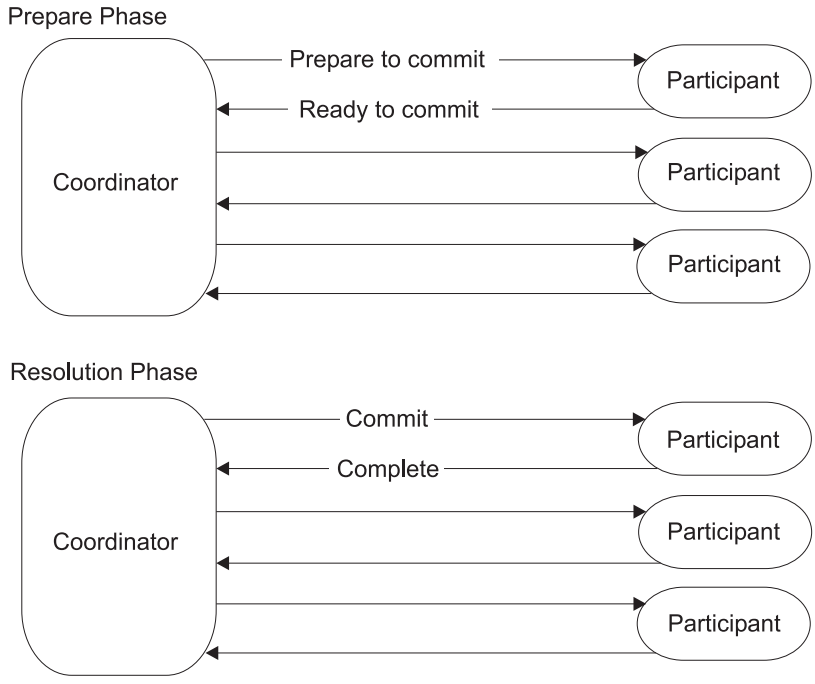

Figure 48. The two-phase commit protocol

The states of a transaction

A transaction progresses through several states during its lifetime. As shown in Figure 49 on page 112, the states associated with the prepare phase are as follows:

- In the *active* state, a transaction may be accessing or modifying data.
- In the *inactive* state, a transaction may be waiting for input from a user.
- In the *preparing* state, a transaction may be preparing to notify a coordinator of its readiness to commit.

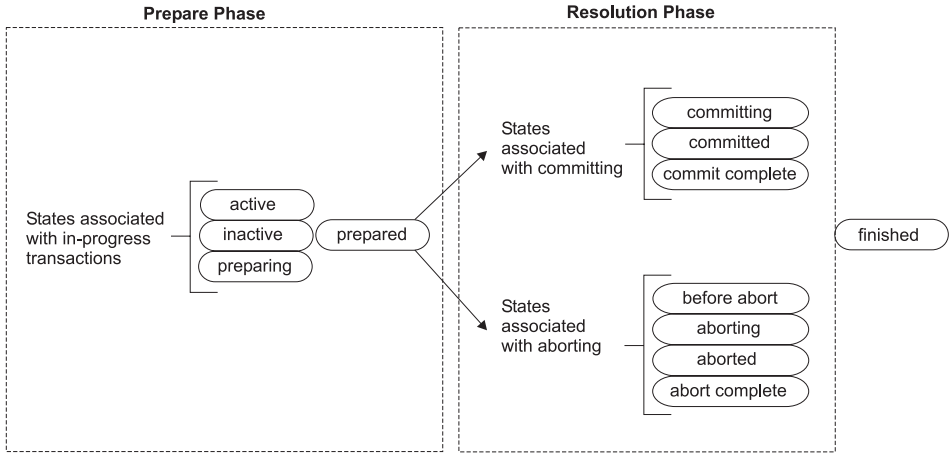

Figure 49. Transaction states

Once a transaction notifies a coordinator of its readiness to commit, the transaction enters the *prepared* state. The next phase is the resolution phase. In the resolution phase, depending on the instruction from the coordinator, a transaction can enter the states in the "committing" or "aborting" track. The final transaction state is *finished*. Note that because a transaction can abort at any time during the prepare phase, the states associated with aborting can occur during that phase; the abort-related states are not repeated in the figure to simplify the illustration.

Note: There are additional transaction states not described here. See the reference page for the **tkadmin list transactions** command for a complete description of all transaction states.

Nested transactions

Encina supports nested transactions. A *nested transaction* is a transaction started within the scope of another transaction. Nested transactions can be used to break a transaction into hierarchies. A transaction that starts another transaction is called a *parent transaction*, and the subtransactions are called *child transactions*. Each *subtransaction* is a discrete set of operations. A subtransaction can in turn begin another subtransaction.

Nested transactions can be aborted without aborting the parent and can therefore be used to isolate failures. For example, a transaction for withdrawing money from a bank account is partitioned into several subtransactions, each representing one subtask in the larger task of withdrawing money. The subtasks consist of individual attempts to withdraw the money. Each "withdrawal" subtask is isolated (using a subtransaction) so

that if the subtask fails, the subtransaction aborts but the parent does not. The parent transaction can then retry the task. Once the task is completed, the parent can commit the transaction.

The relationships among transactions are important in determining whether an entire transaction commits or aborts. A subtransaction is dependent on the outcome of its parent transaction. The state of a subtransaction (known as the *local state*) is visible only to its parent transaction. The parent transaction decides whether to commit or abort based on the individual local states of each child transaction. The parent reports its local state to the coordinator. The coordinator, after polling all participants and arriving at a *global state*, instructs each participant on how to end the transaction—whether to issue a commit or an abort. If a parent transaction commits, all subtransactions also commit. If a parent transaction aborts, all subtransactions also roll back work even if they had issued a local commit.

Refer to the *Encina Transactional Programming Guide* for details on transaction processing.

Transaction identifiers

Encina uses several identifiers to track transactions. The identifiers are as follows:

- A local *transaction identifier* (TID) is an identifier for a transaction in a given server. It is known only to that server.
- A *global TID* ties a transaction to all of its participating applications across different servers. The global TID is unique across all servers.
- An *application ID* identifies a participating application.

For example, Figure 50 on page 114 shows a transaction whose global TID is 87. Application ID 6 originated TID 87; application IDs 7 and 8 are also participants in TID 87. In server 1 on node A, the transaction associated with global TID 87 is known as local TID 27; in server 2 on node B, it is known as local TID 43. The dotted line delimits the participating applications and transactions associated with global TID 87.

The transaction known as local TID 43 in server 1 is different from the transaction known as local TID 43 in server 2 (local TIDs are unique to a server). In server 1, local TID 43 is associated with global TID 64.

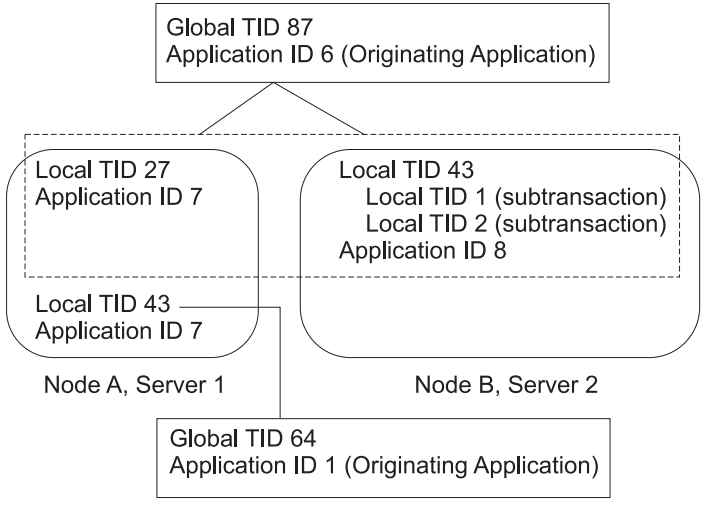

Figure 50. Identifiers used to track transactions across servers

Transactions and locking

Encina transactions use *locking*. Locking enables a transaction to restrict access to resources on which it depends. This ensures that two or more transactions cannot modify the same data at the same time. Locking is important to guarantee the isolation property of transactions. Without locking, data could easily become inconsistent. For example, if two transactions, T1 and T2, require access to the same data, and T1 modifies the data without restricting access to it by other transactions, T2 can then read that data and act upon the value it reads. If T1 subsequently aborts, its modifications to that data are undone; T2 is then acting upon modified data that is no longer accurate.

A transaction requests a lock on a resource to do its work. When the transaction obtains the lock, it is said to be *holding* a lock. When it cannot obtain a lock (possibly because another transaction is holding the lock), it is said to be *waiting* for a lock.

Locks are obtained on behalf of transactions in a given *lock mode*. A lock mode specifies the degree of concurrent access permitted to locked data. A lock requested by a transaction is not granted if an unrelated transaction holds the lock in a mode that does not permit concurrent access. Two lock modes that do not permit concurrent access are referred to as *conflicting*. For example, a transaction cannot obtain a read lock on data for which another transaction holds a write lock. Read and write lock modes are conflicting.

Certain types of locks (such as read locks) are *shared locks*. Shared locks allow multiple transactions to examine data concurrently but not to change it. For example, two transactions can each obtain a read lock on the same data; two read locks are *nonconflicting*.

Lock information can be used to investigate the source of locking conflicts that cause two or more transactions to deadlock. If one transaction holds a lock in a particular mode on an object and a second unrelated transaction requests a lock on the same object in a conflicting mode, the second transaction must wait for the first transaction to release the lock before it proceeds. If the first transaction never releases the lock, the second transaction cannot proceed, and the two transactions are *deadlocked*. This can easily happen if locks are requested out of order. For example, suppose two transactions (transactions 1 and 2) require locks on object A and B to complete. If transaction 1 holds a lock on object A and requests a lock on B, while transaction 2 holds a lock on object B and requests a lock on A, the two transactions are deadlocked. Using lock information from Enconsole and from server-specific commands, you can determine which transaction to terminate to break the deadlock, or you can take another action.

Viewing server transaction information

The Enconsole Server Transactions display screen shows information about the transactions for a server. Choose the **Server Transactions** command from the **View** menu to access this display screen. For each transaction at the selected server, this screen displays the following:

- The local Encina TID
- The number of locks held by the transaction and the number of locks the transaction is waiting for
- The state of the transaction
- The in-progress RPC related to the transaction

Figure 51 on page 116 shows transactions in various states. Note that nested transactions are indented in the list of TIDs.

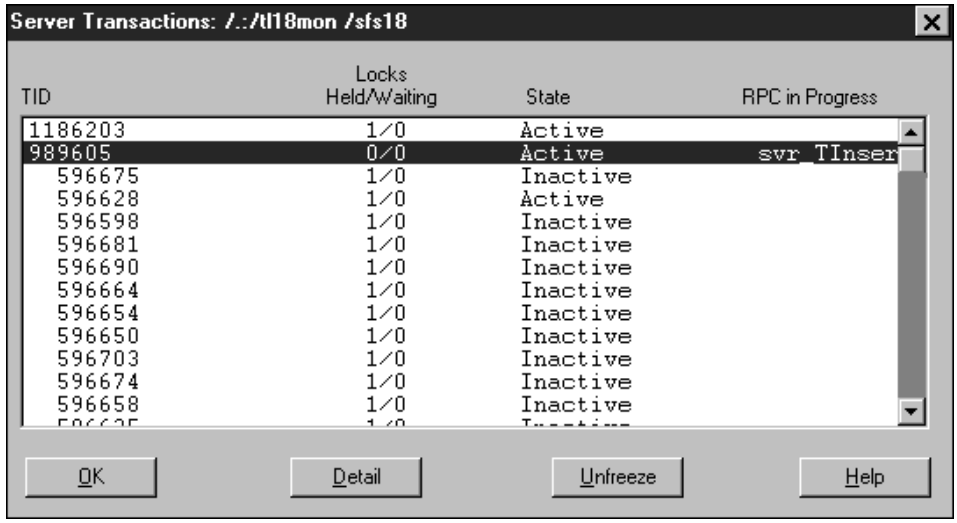

Figure 51. Server Transactions display screen

Enconsole updates the Server Transactions display screen continually unless you choose the **Freeze** button to stop the updates. To cause Enconsole to resume updating the screen, choose the **Freeze** button again.

To review details about a specific TID, select the TID and choose the **Detail** button on the Server Transactions display screen. The Transaction Detail form (Figure 52 on page 117) displays the following information:

- The local Encina TID
- The global TID
- The state of the transaction
- The number of locks the transaction is holding
- The number of locks the transaction is waiting for
- The application IDs of participants in the transaction
- The in-progress RPCs associated with the transaction
- Any subtransactions associated with the transaction

If the transaction being viewed is a subtransaction, this form also lists the following:

- The parent TID
- The application ID of the participant that originated the transaction

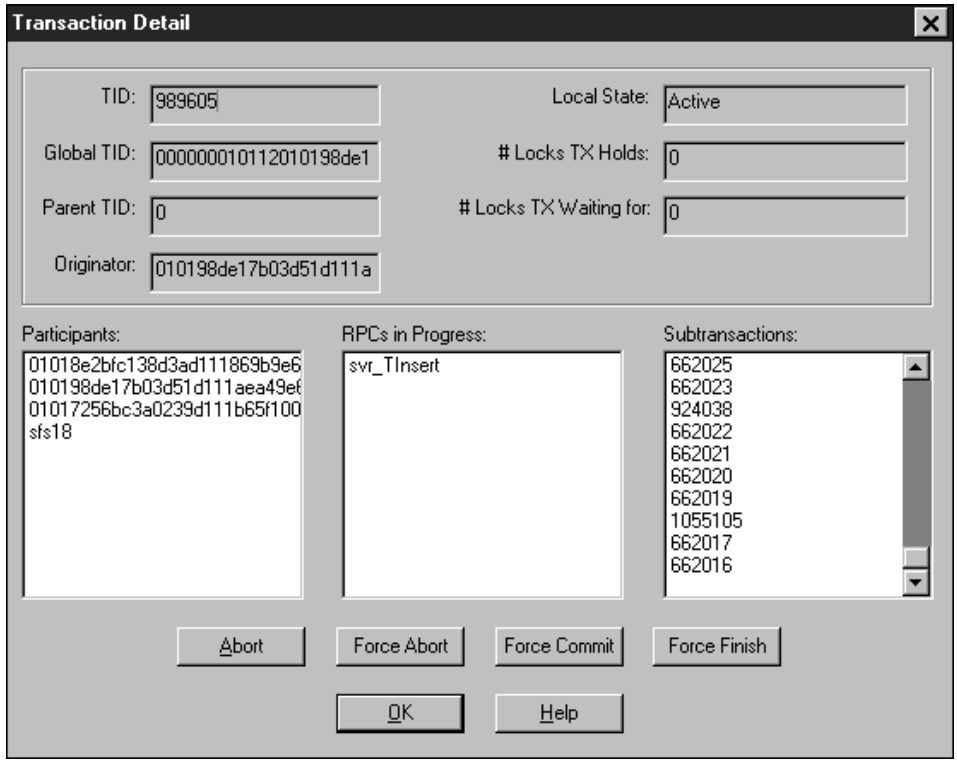

Figure 52. Transaction Detail form

Perform the following steps to review a transaction:

1. Choose the **Server Transactions** command from the **View** menu, and then choose your server from the Selection box. Enconsole displays the Server Transactions display screen.
2. Select a transaction and then choose the **Detail** button. Enconsole displays the Transaction Detail form.
3. On the Transaction Detail form, review the local state and the subtransactions. When you have finished reviewing the transaction details, choose the **OK** button. Enconsole displays the Server Transactions display screen. You can continue to select transactions and review transaction details.

You can also use the following **tkadmin** commands to display information about transactions:

- The **tkadmin list transactions** command, which displays all unresolved transactions and their current states.

- The **tkadmin query transaction** command, which displays details about a transaction's state, initiating application, participating applications, global identifier, and start time.

Administering transactions

Administrators do not normally need to intervene in server transactions. Timeouts associated with transactions prevent any one transaction from holding resources at a server for too long. For example, if two transactions are competing for the same resource (one holds a lock on a resource and the other is requesting that lock, and the lock modes conflict) timeouts will eventually abort one of the transactions: the idle timeout will abort a transaction that is inactive too long, and the operation timeout will abort an active transaction that is taking too long. Nevertheless, a transaction can "hang" indefinitely if, for example, the transaction is prepared but the coordinator is unreachable.

If a hung transaction is interfering with the operation of your server (perhaps holding locks that other critical transactions are waiting for), you may need to intervene. This section describes the types of actions you can take to handle problematic unresolved transactions.

Intervening in unresolved transactions

To investigate why a transaction is unresolved, begin by using the Enconsole Server Transactions display screen to review transaction information at the server. The action that you take for an unresolved transaction depends on the transaction's state, that is, whether or not it has prepared. In general, the following rules of thumb apply:

- If a transaction has not yet prepared, you can abort the transaction at any time.
- If a transaction has prepared, you can allow the system to resolve it (allow normal processing to take place) or you can force an outcome.

Intervening in unprepared transactions

An unprepared transaction can be aborted at any time. Unprepared transaction states include active, inactive, and preparing. Aborting an unprepared transaction does not affect the consistency of data—the transaction's participants have not yet notified a coordinator of their readiness to commit. If you abort an unprepared transaction, any work that was completed on behalf of the transaction is rolled back.

When you abort a transaction, the aborted transaction releases locks that other transactions may be waiting for, possibly freeing a server to perform other operations. However, any work that was done by the transaction must be

redone. You must weigh the impact of aborting a transaction. Allowing the transaction to release its locks (abort) can result in better server performance but with a penalty of later rework.

Intervening in prepared transactions

With transactions that have already prepared, any action that you take is more significant. To release the locks held by a prepared transaction, you must force it to commit or to abort. Forcing might be required in the following situations:

- When a transaction can never be completed
- When critical resources are held
- When system performance suffers

You can introduce data inconsistencies whether you force a transaction to commit or force it to abort. Forcing a transaction to abort results in better damage reporting to other applications.

Evaluating transactions

You can examine the following areas to determine whether to force a transaction to commit or to abort:

- *Review the transaction state in each of the transaction's participating applications*. Participating applications of a transaction share a unique global TID. In some cases, you can determine the intended outcome of a transaction by reviewing the local transaction states of each participant.
- *Examine locks*. A hung transaction can be caused by conflicting locks. After using the Transaction Details Screen to determine the number of locks held and waiting, you may need additional information about a lock's mode. You can obtain information about the types of locks being held by a particular transaction by using server-specific lock commands. For example, the following commands can be used to investigate the source of locking conflicts that can cause two or more transactions to deadlock:
  - The **sfsadmin query tranlock** command (for SFS servers). This command displays the lock mode that the transaction holds, the name of the file on which the lock is placed, and the key value that is locked.
  - The **rqsadmin list locks** command (for RQS servers). This command displays information about all locks held or requested (waiting) in a server. It can also be used to view locks for a specified transaction, queue, or queue set.

Intervening in problematic transactions

Note: You can introduce inconsistencies into server data by forcing a transaction to the wrong outcome.

Perform the following steps to intervene in a problematic transaction:

1. Review the Transaction Detail form.

2. Abort the transaction or force an outcome by choosing the appropriate button on the Transaction Detail form. The following options are available:
   - **ABORT** — Aborts a transaction that is *not yet in the prepared state*. When you abort an unprepared transaction, there is no risk to the consistency of the transaction data. Operations that the transaction completed are undone.
   - **FORCE ABORT** — Releases all locks held *by a prepared transaction* and forces the transaction to abort.
   - **FORCE COMMIT** — Releases all locks held *by a prepared transaction* and forces the transaction to commit.
   - **FORCE FINISH** — Forces a transaction to finish regardless of whether its outcome has been reported to all participating applications. Normally, a transaction must report its outcome to all participating applications as a requirement of the transaction's successful completion. All local prefinish work must be done; specifically, the resolution phase must be completed. The **FORCE FINISH** option is useful if one of the participating applications has become permanently unavailable.

You can also use the following **tkadmin** commands to manage transactions:

- The **tkadmin abort transaction** command, which aborts a transaction that is not yet in the prepared state. This command is the equivalent of the **ABORT** button in Enconsole.
- The **tkadmin force transaction** command, which forces a transaction outcome. If this command is issued with no options, the transaction is forced to abort. This command is the equivalent of the **FORCE ABORT** button in Enconsole. The **-commitdesired** option forces the transaction to commit. Using this option is the equivalent of the **FORCE COMMIT** button in Enconsole. The **-finish** option forces a transaction to finish regardless of whether its outcome has been reported to all participating applications. Using this option is the equivalent of the **FORCE FINISH** button in Enconsole.

Chapter 6. Performing backups

This chapter describes the procedures used to back up Toolkit server log volumes, data volumes, and restart data. It also provides procedures for managing backup files.

Overview

You can use Encina's backup facility to back up server data for Encina recoverable servers such as cell managers, Structured File Server (SFS) servers, Recoverable Queueing Service (RQS) servers, or any server managing data in Toolkit volumes. In the event of a media failure, server data can be restored from Encina backup files, log archive files, and the log file as described in "Chapter 7. Recovering from failures" on page 133.

The Encina recovery process uses the following files:

- The *log file* stores information about updates made to recoverable data. A log file can be thought of as a running journal of changes made to application data. A recovery service uses the information stored in a log file to restore modified data to a consistent state in the event of a system failure. *The log file is required to restore the volume to its latest consistent state.* The log volume must be mirrored to ensure the availability of the log file. See "Chapter 3. Creating volumes" on page 69 for information on mirroring.
- *Log archive files* back up log data in a server's log volume. Log archive files are automatically generated when media archiving is enabled. Media archiving moves older data from a log file to log archive files to free storage space. The Encina recovery process applies the changes recorded in log files and log archive files to backup files to restore a data volume to a consistent state. (The log archive files are sometimes referred to as media recovery archive (MRA) files in Encina messages.)
- *Backup files* back up application data in a server's data volume. A complete backup (consisting of one or more backup files) covers an entire volume. Backup files must be created manually with the **tkadmin backup lvol** command.

  Encina associates each complete backup with the particular log archive file that contains the earliest data required to restore the volume. (This earliest log archive file marks the beginning of a sequence of archive files required for that backup.) A complete backup, along with its associated log files, is required to restore a data volume.

Figure 53 shows the use of backup files, log archive files, and the log file in the recovery process. Backup files alone are insufficient to restore a volume. Backup files *and* log information restore data to a consistent state. Backup files hold a copy of the volume as it stood when the last backup was made; log information brings the old data forward (applying the more recent changes to data that had not yet been backed up). As shown in the figure, older log information is stored in the archives; current information is in the log file.

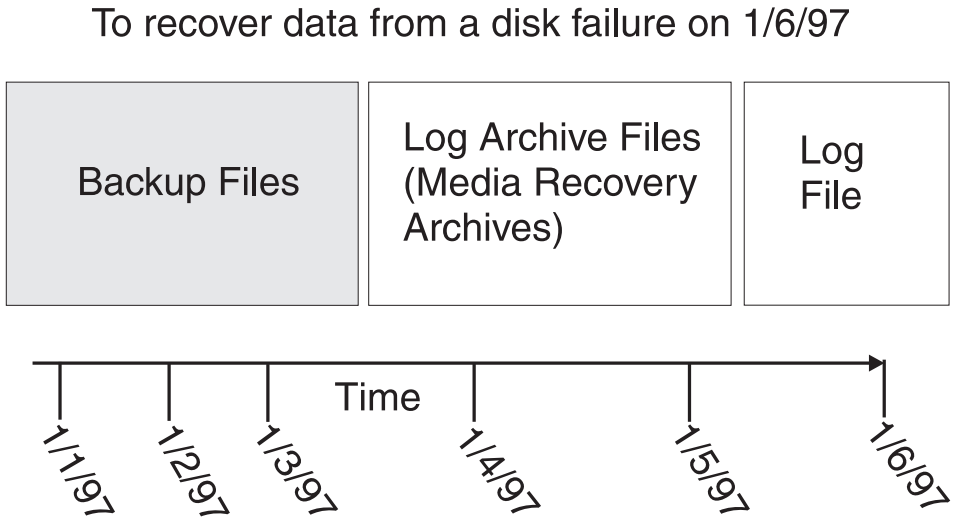

Figure 53. Role of backups in recovery

The first time you start a server, a file named **restart** is automatically created in the server's working directory. This file contains volume names, locations, and configurations. On subsequent restarts, the server uses this file to recover configuration information about volumes. Server restart files must be manually backed up by using operating system commands.

The following steps summarize the procedure you follow to create and manage backups for a server:

1. Enable media archiving — To generate log archive files and back up data volumes, you must enable media archiving. See "Media archiving" on page 123 for details.
2. Back up server data volumes — Use the **tkadmin backup lvol** command to back up server data volumes. See "Backing up server data volumes" on page 124 for details.
3. Back up server restart data — In addition to backing up server volumes, you must also back up server restart files. See "Backing up server restart data" on page 128 for details.

4. Move backups offline — To minimize the risk of losing data, you must move log archive, backup, and restart files offline. See “Moving backups offline” on page 128 for details.
5. Manage backup files — You must periodically remove old backups to reclaim storage space. Use the **tkadmin query backup** command to obtain information about backups and the **tkadmin retain backups** command to retain information about a certain number of backups. See “Managing backup files” on page 129 for details.

Media archiving

Media archiving is the process of transparently moving older data from a log file to log archive files to free storage space. Media archiving must be enabled to automatically generate the log files needed to recover a server’s data volumes. This section describes how to enable media archiving and, if necessary, how to flush the media archive (write cached log records from a log file to log archive files).

Enabling media archiving

To generate log archive files and create backups, you must enable media archiving. Media archiving can be enabled for servers that manage data in Toolkit volumes (cell managers, SFS servers, and RQS servers), but not for servers that store only log data (node managers and Peer-to-Peer Communications (PPC) gateway servers).

With media archiving enabled, log archive files are automatically generated and, by default, stored in the **logArchive** directory under the server’s working directory. You must allocate adequate disk space (approximately 4 MB) for this directory. Log archive filenames are based on the name of the log volume storing the server’s log file and a sequence number beginning with 0 (zero). For example, an RQS server that stores its log file on a volume named **rqslogvol** creates archive files named **rqslogvol.LA.0**, **rqslogvol.LA.1**, **rqslogvol.LA.2**, and so on.

Use the **tkadmin query mediaarchiving** command to determine whether media archiving is enabled for a server. The syntax for the command follows:

```
tkadmin query mediaarchiving -server server_name
```

Use the **tkadmin enable mediaarchiving** command to enable media archiving. The syntax for the command follows:

```
tkadmin enable mediaarchiving -server server_name
```

Perform the following procedure to enable media archiving:

1. Use the **tkadmin query mediaarchiving** command to determine whether media archiving is enabled for a server. For example, enter the following command to check media archiving in a server:

   ```
   % tkadmin query mediaarchiving
   Media archiving is disabled.
   ```

2. Use the **tkadmin enable mediaarchiving** command to enable media archiving for a server. For example, enter the following command to enable media archiving:

   ```
   % tkadmin enable mediaarchiving
   ```

To minimize the risk of losing data, you must move log archive files offline as soon as they are completed. See "Moving backups offline" on page 128 for details.

Flushing the media archive

A server intermittently writes data to log archive files. If a backup is done but there is insufficient activity to trigger writing to the log archive, information about the recent backup is not reflected in the archive file. You can use the **tkadmin flush mediaarchive** command to ensure that you have captured all recent activity in the log archive files. This command writes the cached data to the active archive file, closes the file, and opens a new archive file for recording subsequent log activity. The syntax is as follows:

```
tkadmin flush mediaarchive -server server_name
```

The following command flushes log records from the log file to log archive files:

```
% tkadmin flush mediaarchive
```

Backing up server data volumes

Backing up a data volume involves dumping its contents to a set of backup files by using the **tkadmin backup lvol** command. The command backs up the volume; it does not back up data structures such as files or queues. To use the **tkadmin backup lvol** command, media archiving must be enabled (see "Media archiving" on page 123). You can back up a volume while the server is running, although performance may be slowed.

Each backup file covers some part of the volume. A complete backup exists when all of the volume has been copied once. Backups are incremental—after a complete backup of a volume is created, each additional backup file ends a new complete backup. Different backups of the same volume can overlap or share files. Figure 54 on page 125 illustrates how complete backups can overlap.

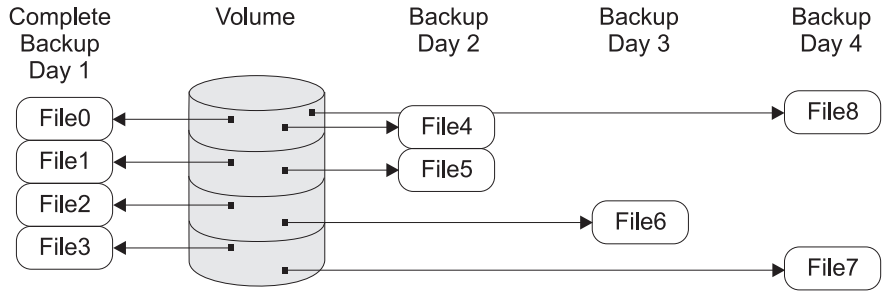

Figure 54. Multiple complete backups

On day 1, a complete backup consisting of four files, 0 through 3, is made. On day 2, two additional backup files are made, each creating a new complete backup (files 1 through 4 and 2 through 5). By day 4, two independent complete backups exist (files 0 through 3 and 4 through 7). Two backups are independent if they do not overlap or share files.

Backup file names have the form *file_prefix*.**TRB.***sequence_number*. The *file_prefix* determines the location of the file in the native file system. The string **TRB** is a constant that identifies backup files. The *sequence_number* is appended to make the backup file names belonging to a particular volume unique. The sequence numbers are automatically generated by the system. Complete backups are identified by the sequence number of the *most recent* file in the backup. For example, the first complete backup, which ends with file 3, is considered backup 3 (not the third backup).

Using the tkadmin backup lvol command

The syntax of the **tkadmin backup lvol** command follows:

```
tkadmin backup lvol -server server_name logical_volume_name \
[-fileprefix file_prefix]
[-filesize file_size] \
[-nfiles number_of_files]
```

Specify the name of the server for the *server_name* argument and the name of the logical volume to be backed up for the *logical_volume_name* argument.

By default, this command generates one backup file containing at most 1 MB of information. If your volume is larger than 1 MB, using the default values results in a *partial* backup. Recall that a complete backup exists when all of the volume has been copied once. If more than 1 MB of data needs to be backed up, the **tkadmin backup lvol** command returns the following warning:

```
As yet, there is no complete backup of volume volume_name
```

You must reissue the **tkadmin backup lvol** command (which generates an additional 1-MB backup file each time it is issued) until no warning is returned. Alternatively, you can use the **-filesize** and **-nfiles** options to specify the size and number of backup files needed to back up the entire volume. See "Creating a complete backup" for an explanation of how to calculate the required size and number of files.

The **tkadmin backup lvol** command has the following optional arguments:

- *file_prefix* — Specifies a prefix used for each backup filename. The prefix can be a complete or relative pathname used to write the files to a different location—for example, **/bkups/sfsvol1** (UNIX), **D:\BKUPS\SFSVOL1** (Windows NT), or **sfsvol1**. Using the latter example, the backup files are named **sfsvol1.TRB.000000**, **sfsvol1.TRB.000001**, **sfsvol1.TRB.000002**, and so on. A relative pathname is interpreted within the context of the server's working directory. If a file prefix is not specified, the default is the volume name.
- *file_size* — Specifies the file size. The *file_size* argument can be specified in bytes (an integer) or in kilobytes (an integer followed by the letter **k**—for example, **51k**). If a file size is not specified, the default file size is 1 MB.
- *number_of_files* — Specifies the number of files to be created. If a number of files is not specified, the default is one file.

Creating a complete backup

You can create a complete backup by reissuing the **tkadmin backup lvol** command until no warning is returned, or by specifying the size and number of backup files needed to back up the entire volume. The Encina Recovery Service keeps track of the backup files created and automatically begins each new backup file where the last one ended. A complete backup exists when enough files are created to restore the entire volume.

To create a complete backup, perform the following steps:

1. Determine the size and number of backup files you want to create. By default, one backup file containing at most 1 MB of information is generated. This may not be sufficient for large volumes.

   If a volume is very large, you can specify a file size and number of files sufficient to create a complete backup with one or two issues of the command. Use the **tkadmin query lvol** command to determine the volume size. The command syntax follows:

   ```
   tkadmin query lvol -server server_name logical_volume_name
   ```

   Specify the name of the data volume for the *logical_volume_name* argument. For example, enter the following command to display information for a volume named **sfsvol1**:

   ```
   % tkadmin query lvol sfsvol1
   ```

```
Information about logical volume sfsvol1
All sizes and offsets are in pages. Page size is: 4 Kbytes
size: 1998 chunksize: 64
Backing physical volumes (only 'clean' and 'dirty' volumes\
are active):
sfsvol1_physicalVol1 (clean)
Volume is currently enabled.
```

As the output indicates, the volume is 1998 pages (7992 KB) in size. You must allow 1 KB per backup file for header information. Therefore, to make a complete backup for this volume, each backup file must be (7992/*n*) + 1 KB in size, where *n* is the number of files to be created. For example, to create a complete backup consisting of eight files, each file must be 1000 KB in size.

2. Use the **tkadmin backup lvol** to create the backup. Enter the following command to back up a 7992-KB volume named **sfsvol1** with eight backup files of 1000 KB each:

   ```
   % tkadmin backup lvol sfsvol1 -filesize 1000k -nfiles 8
   ```

   The command produces eight backup files, **sfsvol1.TRB.000000** through **sfsvol1.TRB.000007**, each 1000 KB in size. These files constitute a complete backup of the entire volume. This backup is referred to by the backup sequence number **7** (the number of the last file in the sequence). Additional backups result in backup files named **sfsvol1.TRB.000008**, **sfsvol1.TRB.000009**, and so on. Each additional file is a valid endpoint of a complete backup; backup **8** consists of files **1** through **8**, and backup **9** consists of files **2** through **9**.

3. Use the **tkadmin flush mediaarchive** command to immediately write cached log data to an archive file. (The log volume intermittently writes data to archive files. If a backup is done but there is insufficient activity to trigger writing to the log archive, information about the recent backup is not reflected in the archive file.) The syntax is as follows:

   tkadmin flush mediaarchive -server *server_name*

   The following command writes cached log records to the log archives:

   ```
   % tkadmin flush mediaarchive
   ```

Only one backup can be in progress for any given volume. If you try to back up a volume for which a backup is already in progress, the backup in progress is aborted and the original invocation of the backup command returns with an error. If a backup is interrupted by a system or server crash, you can continue the backup after the server is restarted by reissuing the **tkadmin backup lvol** command with the same parameters as before. All previously written files are valid, but the incomplete file is rewritten.

To minimize the risk of losing data, you must move backup files offline as soon as they are completed. See “Moving backups offline” for details.

Backing up server restart data

In addition to backing up server volumes, it is also important to back up server restart data. The first time you use Enconsole to start a server, a file named **restart** is automatically created in the server’s working directory. This file contains information required to restart the server. A copy of the restart file, named **restart.bak**, is stored in the server’s mirror working directory. You must back up the restart file and store the backup offline in case both online copies are lost; see “Moving backups offline” for details.

Moving backups offline

To minimize the risk of losing data, you should move all backups offline as soon as they are completed. This includes all backup files, log archive files, and copies of server restart files.

Use an operating system utility (for example, the **tar** command on UNIX operating systems and the Backup program on Windows NT) to move all backups offline to secure storage. The rate at which you must move backups offline depends on how frequently they are created.

- Log archive files — The rate at which log archive files are generated depends on how much log data is generated by the server. The log archive file with the highest sequence number is the most recent archive file. It is possible that the server is still writing data into that archive file and it should not be moved offline. Move only inactive (complete) archive files offline.

  Note that you can use the **tkadmin flush mediaarchive** command to immediately write cached log data to an archive file. This writes the cached data to the active archive file, closes the file, and opens a new log archive file for recording subsequent log activity. See “Creating a complete backup” on page 126 for details on this command.

  Log archive files should be moved offline *in duplicate*. Each complete backup is associated with a log archive file. This archive file is the earliest file needed to restore a volume to a consistent state. (It marks the beginning of a sequence of archive files required for that backup.) To successfully restore a volume, you need the entire sequence of archive files.
- Backup files — The frequency with which you back up a server’s data volumes depends on how much log data you are willing to retain. Each complete backup of a server’s data volume is associated with a particular log archive file that contains the earliest log data required to restore the volume. The older the backup, the more log archive files are required to

restore the volume. Use the **tkadmin query backup** command to determine the earliest log archive file required by a particular backup (see "Querying a backup of a data volume" for details). Move backup files offline as soon as they are completed.

- Restart files — All administrative commands affecting a server's volume configuration change the restart file. For example, creating new logical volumes for the server or renaming the server's existing volumes changes the restart file. To ensure you maintain the latest copy of a server's restart file, back up the restart file once after the server is initially configured and each time you make changes to the server's configuration. Move the backup file offline as soon as it is completed.

To restore a volume or server restart file, you must move the appropriate files back online as described in "Chapter 7. Recovering from failures" on page 133.

Managing backup files

To reclaim storage space, you must determine how many backups to retain and periodically remove old backup and log archive files. Use the **tkadmin query backup** command to obtain information on backups and the **tkadmin retain backups** command to retain information about a specified number of backups and invalidate information for old backups.

Querying a backup of a data volume

Use the **tkadmin query backup** command to obtain information on backups of a data volume. This command determines how many complete, independent backups exist for a data volume and the earliest log archive file required for each complete backup. The command displays the names of the files constituting a backup, the date and time each file was created, and the earliest log archive file required for the backup (if a complete backup).

The syntax of the **tkadmin query backup** command follows:

```
tkadmin query backup -server server_name logical_volume_name
[-backupfilenum backup_file_number] [-count count] [-all]
```

You must specify the logical volume name for the *logical_volume_name* argument. You can specify the following options and arguments with the command:

- **-backupfilenum** *backup_file_number* — Specifies the backup file number of the backup to display. The backup file number is the sequence number of the file that ends the complete backup. The *backup_file_number* argument defaults to the most recent backup for the specified volume.
- **-count** *count*— Specifies the number of complete, independent backups to display, starting with the most recent backup.
- **-all** — Displays a list of all complete, independent backups for the volume.

The following sections provide examples of how the **tkadmin query backup** command can be used to display information about backups.

Querying the most recent backup

To display information about the most recent complete backup for a volume, use the **tkadmin query backup** command without specifying any options. For example, enter the following command to display information for a volume named **rqs1_dataVol**:

```
%  tkadmin query backup rqs1_dataVol

A backup for volume rqs1_dataVol comprises 6 files:
In directory . (relative to server's working directory) :
        rqs1_dataVol.TRB.000012 (started Fri Sep 12 09:55:22 1997)
        rqs1_dataVol.TRB.000013 (started Fri Sep 12 09:55:24 1997)
        rqs1_dataVol.TRB.000014 (started Fri Sep 12 09:55:25 1997)
        rqs1_dataVol.TRB.000015 (started Fri Sep 12 09:55:27 1997)
        rqs1_dataVol.TRB.000016 (started Fri Sep 12 09:55:28 1997)
        rqs1_dataVol.TRB.000017 (started Fri Sep 12 09:55:29 1997,
end of backup 17)
Earliest log archive file required to restore this data volume from
this backup (without restoring associated log volume):
/opt/encinalocal/trader/server/rqs1/logArchive/rqs1_logVol.LA.43

 Earliest log archive file required to restore the log volume can
 be determined via the tkadmin query logvol command.
```

As the output indicates, the most recent backup of **rqs1_dataVol** consists of six backup files, numbered **12** through **17**. The log archive file named **logvol.LA.43** is the earliest log archive file required to restore the volume using this backup.

Querying a specific backup

To display information about a specific backup, use the **-backupfilenum** option with the **tkadmin query backup** command. For example, enter the following command to display information about an older backup ending with file number **3** for a volume named **sfsvol1**:

```
%  tkadmin query backup -backupfilenum 3

A backup for volume sfsvol1 comprises 4 files:
In directory . (relative to the server's working directory):
    sfsvol1.TRB.000000 (started Tue Jan 2 18:01:43 EST 1996)
    sfsvol1.TRB.000001 (started Tue Jan 2 18:14:32 EST 1996)
    sfsvol1.TRB.000002 (started Tue Jan 2 18:21:31 EST 1996)
    sfsvol1.TRB.000003 (started Tue Jan 2 18:28:32 EST 1996,
end of backup 3)

Earliest log archive file required to restore this data volume
 from this backup (without restoring associated log volume):
/opt/encinalocal/branch1/servers/sfs1/logArchive/logvol.LA.39

Earliest log archive file required to restore the log volume can
 be determined via the tkadmin query logvol command.
```

The backup that ends with backup file number **3** includes files numbered **0** through **3**. The log archive file named **logvol.LA.39** is the earliest log archive file required to restore the volume using this backup.

Listing all complete, independent backups

To display information about all complete, independent backups for a volume, use the **-all** option with the **tkadmin query backup** command. For example, enter the following command to display information for a volume named **sfsvol1**:

```
%  tkadmin query backup sfsvol1 -all
Volume sfsvol1 contains the following complete backups.
(The most recent backup appears first.)
1: The first complete backup comprises the following 4 files:
  In directory . (relative to server's working directory):
    sfsvol1.TRB.000007 (started Tue Jan 3 19:35:30 EST 1996)
    sfsvol1.TRB.000006 (started Tue Jan 3 19:28:32 EST 1996)
    sfsvol1.TRB.000005 (started Tue Jan 3 19:21:31 EST 1996)
    sfsvol1.TRB.000004 (started Tue Jan 3 19:14:32 EST 1996)
  Earliest MRA file required to restore this data volume from this backup:
/opt/encinalocal/branch1/servers/sfs1/logArchive/logvol.LA.74
2: The second complete backup comprises the following 4 files:
  In directory . (relative to server's working directory):
    sfsvol1.TRB.000003 (started Tue Jan 2 18:28:32 EST 1996)
    sfsvol1.TRB.000002 (started Tue Jan 2 18:21:31 EST 1996)
    sfsvol1.TRB.000001 (started Tue Jan 2 18:14:32 EST 1996)
    sfsvol1.TRB.000000 (started Tue Jan 2 18:01:43 EST 1996)
  Earliest MRA file required to restore this data volume from this backup:
/opt/encinalocal/branch1/servers/sfs1/logArchive/logvol.LA.39
```

The output displays two independent backups. The first (and most recent) independent backup consists of backup files **4** through **7**. The second independent backup consists of backup files **0** through **3**.

Retaining backups

Use the **tkadmin retain backups** command to keep information about a specified number of complete, independent backups, starting with the most recent backup. The command does not remove any backups, but it invalidates the information associated with old backups. *The command deletes any log archive files associated with the invalidated backup files.* Upon successful completion, the command displays a list of backup files that can be removed. Use an operating system command to remove the files.

The command syntax follows:

tkadmin retain backups -server *server_name logical_volume_name number_of_backups*

Specify the name of the logical volume for the *logical_volume_name* argument and the number of independent backups to retain for the *number_of_backups* argument. For example, assume the volume **sfsvol1** has three independent backups: one consisting of files **0** through **3**, another consisting of files **4**

through **7**, and the third consisting of files **8** through **11**. Enter the following command to retain the two most recent independent backups for the volume named **sfsvol1**:

```
%  tkadmin retain backups sfsvol1 2

Truncated backup for volume sfsvol1
The following files are no longer needed.
numFiles: 4
     0: sfsvol1.TRB.000000
     1: sfsvol1.TRB.000001
     2: sfsvol1.TRB.000002
     3: sfsvol1.TRB.000003
```

The output indicates that files **0** through **3** can be removed safely.

Querying a log volume

Use the **tkadmin query logvol** command to obtain information about a log volume. The command displays the following information:

- The name of the log volume.
- The name of the associated archive device.
- The name of the oldest archive file stored on that device.
- The available free space on the log volume.
- The number and names of the log files stored on the volume.

The syntax of the **tkadmin query logvol** command follows:

tkadmin query logvol -server *server_name logical_volume_name*

The following example displays information about the log volume **rqs1_logVol**:

```
%  tkadmin query logvol rqs1_logVol

Information about log volume : rqs1_logVol
         Archive device :
    /opt/encinalocal/trader/server/rqs1/logArchive
         Oldest offline archive file :
    /opt/encinalocal/trader/server/rqs1/logArchive/rqs1_logVol.LA.43
         Number of free pages : 2048
         Number of log files : 1
         Log file list :
                 logFile
```

Chapter 7. Recovering from failures

This chapter provides instructions for recovering from the following types of Encina server failures:

- *Abnormal Server Termination*—An abnormal server termination is a configuration or system problem that causes a server to terminate unexpectedly. For example, a power failure can cause a server to terminate abnormally.
- *Media Failure*—A media failure is any failure of disk storage that causes a loss of data. For example, a disk head crash that makes the disk unavailable or destroys data is a media failure. Media failures can affect any of a server's logical volumes (a log volume or a data volume). The procedure to recover from a failure depends on the type of volume affected, the type of device backing the logical volume, and your backup and archiving policies.

Abnormal server termination

A Toolkit server can terminate abnormally due to system problems. At the time of failure, active transactions are interrupted. The system automatically recovers from this type of interruption when the server is restarted in normal operations mode. Specifically, the server's log file is replayed to update the server's data to its state just before the time of failure.

Toolkit server data is transactionally consistent across restarts. In most cases, restarting the server is all that is needed to fix the problem. See "Starting servers" on page 63 for more information on restarting a server.

Media failure

Physical storage media can fail for many reasons. One common cause of media failure is a hardware (disk) failure. When a media failure occurs, you must determine the extent of the damage and restore a server's data before resuming server operation.

Diagnosing a media failure

When a media failure occurs and a server's attempt to use a disk fails, the server issues a warning message. An example warning message follows:

```
"Volume sfslvol1 error 0x7857B004 disk /dev/rsd3c
VolPageNumber 0x00000001 DiskPageNum 0x00000008"
```

Server messages are sent through the Encina Trace Facility to a specific destination. The default destination for **error** trace class output (which

includes warning messages) is the server's standard error stream, or the Serious Messages display screen if you are using Enconsole.

After reviewing the Encina error message, check the operating system error log for further information on the disk failure. For example, on UNIX, view all error output to determine the following:

- The name of the damaged disk
- The disk sector that is damaged
- The name of the logical volume that is backed by the damaged disk
- The availability of the volume's data (whether the volume was mirrored and whether at least one mirror remains undamaged)

You can use Enconsole (or the **tkadmin query lvol** command) to determine exactly how a logical volume was affected by a media failure. Depending on the extent of the damage, you can choose the appropriate repair procedure. You need to either repair a failed mirror or restore a data or log volume.

Viewing restart files

When a server is not available or cannot be started, you can use restart information from Encina volume restart files to determine how a server's volumes are configured. This information is an alternative to issuing the **tkadmin** administrative commands that display volume and mirror information (for example, the **tkadmin query lvol** and **tkadmin query pvol** commands).

Restart files should be read for informational purposes only. Any attempt to modify them manually can lead to irreversible damage to the restart data. It is recommended that you contact your product support representative for assistance in evaluating and resolving volume configuration problems. If your volumes are managed by the AIX LVM instead of by Encina, use AIX utilities to determine how volumes are configured.

A *volume restart file* contains volume names and their disk locations. Restart files are automatically created when the server is started for the first time. On subsequent restarts, the server uses the file to recover configuration information about volumes. If you used Enconsole or **enccp** to start a server, the default locations for the restart files of recoverable servers are the following:

- The file **restart** in the server's working directory (for example, **/opt/encinalocal/***monitor_cell***/server/***server_name***/restart** on most UNIX platforms and on Windows NT systems).
- The file **restart.bak** in the server's mirror working directory (for example, **/opt/encinamirror/***monitor_cell***/server/***server_name***/restart.bak** on most UNIX platforms and on Windows NT systems).

For cell managers, the default paths are as follows for most UNIX platforms and for Windows NT systems:

- **/opt/encinalocal/***monitor_cell***/ecm/restart**
- **/opt/encinamirror/***monitor_cell***/ecm/restart.bak**

For node managers, the default paths are as follows for most UNIX platforms and for Windows NT systems:

- **/opt/encinalocal/***monitor_cell***/node/***node_name***/restart**
- **/opt/encinamirror/***monitor_cell***/node/***node_name***/restart.bak**

If you started your server from the command line by using a server-specific command such as **sfs** or **rqs**, or by using an Encina restart script, the restart files were specified as the argument to the **-v** option of the startup command. For cell managers and node managers started with restart scripts, you can view the contents of the restart shell scripts (on UNIX systems) or the command files (on Windows NT systems). On UNIX, the cell manager restart script is **rc.encina.cell**; the node manager restart script is **rc.encina.***node_name*. On Windows NT, the cell manager command file is **encina.cell.cmd**; the node manager command file is **encina.***node_name***.cmd**.

Use operating system commands to view restart files—for example, use **cat** or **more** on UNIX systems or **notepad** on Windows NT. The following example displays volume restart information for an Encina server on a UNIX system. The command is issued in the directory **/opt/encinalocal/trader/server/rqs1**, the working directory of the server.

```
% cd /opt/encinalocal/trader/server/rqs1
% more restart
```

The output consists of lists of attribute-value pairs. The sections of output are as follows:

- Lists that begin `"EntryType" "disk"` provide general information about disks, including raw disk device name, disk size, regions, and offsets.
- Lists that include `"EntryType" "physicalVol"` describe the physical volumes for the server. Information includes the volume name, its raw disk device name, its ID, its state, number of pages, and number of regions.
- Lists that include `"EntryType" "logicalVol"` describe the logical volumes for the server. Information includes the volume name, number of pages, and the ID number of backing physical volumes.

The example output indicates that there are two logical volumes, **rqsLogLvol** and **rqsLvol**. The logical volume **rqsLogLvol** is 992 pages. It has one backing physical volume: **rqsLogPvol** (disk partition **/dev/rdsk/c0t3d0s6**). The logical volume **rqsLvol** is 992 pages. It has one backing physical volume: **rqsPvol** (disk partition **/dev/rdsk/c0t3d0s7**).

```
("LastId" 14
 "ReleaseNumber" 3
 "ReleaseVersion" 0)("EntryType" "disk"
 "Label" "/dev/rdsk/c0t3d0s6"
 "Name" "/dev/rdsk/c0t3d0s6"
 "NumRetries" 5
 "RegionOffset" [0]
 "RegionSize" [992]
 "Size" 1024)("EntryType" "disk"
 "Label" "/dev/rdsk/c0t3d0s7"
 "Name" "/dev/rdsk/c0t3d0s7"
 "NumRetries" 5
 "RegionOffset" [0]
 "RegionSize" [992]
 "Size" 1024)("ChunkSize" 32
 "Disks" ["/dev/rdsk/c0t3d0s6"]
 "EntryType" "physicalVol"
 "Id" 11
 "Name" "rqsLogPvol"
 "NumPages" 992
 "PageNum" [0]
 "RegionOffset" [0]
 "State" 84)("ChunkSize" 32
 "Disks" ["/dev/rdsk/c0t3d0s7"]
 "EntryType" "physicalVol"
 "Id" 13
 "Name" "rqsPvol"
 "NumPages" 992
 "PageNum" [0]
 "RegionOffset" [0]
 "State" 84)("Destroyed" 0
 "EntryType" "logicalVol"
 "Id" 12
 "LvmMapped" 0
 "Mirrors" [11]
 "Name" "rqsLogLvol"
 "NumPages" 992)("Destroyed" 0
 "EntryType" "logicalVol"
 "Id" 14
 "LvmMapped" 0
 "Mirrors" [13]
 "Name" "rqsLvol"
 "NumPages" 992)
```

Restoring lost or damaged data

The procedure to use for restoring lost or damaged data depends on whether your volumes are mirrored and on the extent of the media damage, as follows:

- If the logical volume affected by a media failure was mirrored and a mirror copy of the volume remains undamaged, you can synchronize the failed mirror with the functional one. Data mirroring makes the repair process less costly because the repair can be done without having to restore data.

(Restoring data requires a temporary server outage.) See "Repairing a failed mirror" on page 142 for instructions on repairing a failed mirror.

- If the logical volume affected by the media failure was not mirrored or if the media failure affected all mirror copies, you must stop the server and restore the data by using backup files and log archive files. See "Restoring a data volume" on page 143 or "Restoring a log volume" on page 147 for instructions on restoring a data volume or log volume.

"Ensuring that media Is functional" contains instructions for ensuring that a logical volume is backed by functional disks or a functional AIX logical volume. This is the one common procedure required in all repair or restore instructions. It is explained once, in detail, and referred to throughout the remainder of the repair and restore procedures.

Ensuring that media Is functional

Before you can repair a failed mirror or restore a logical volume, you must first make sure that the disk (or AIX logical volume) backing the failed mirror or logical volume is functional. In some cases, a temporary error in writing to a disk causes a failure. In other cases, the failure is caused by a permanent condition—for example, a hardware malfunction. You must determine the cause and nature of the failure: temporary or permanent.

If the failure is temporary—for example, a power failure—you can repair or restore the server's data when power returns. This can be as simple as restarting the server or issuing a command to instruct the server to synchronize its mirrors. If the failure is permanent, you must first relocate the volume onto functional physical storage devices before repairing or restoring the server's data.

UNIX and Windows NT. You can ensure that a volume is backed by functional disks in one of the following ways:

- Move the physical volume from damaged disks to functional disks (when there is at least one good mirror for the logical volume). See "Moving a physical volume to functional disks (UNIX and Windows NT)" on page 138.
- Recreate the logical volume using different, functional disks (when there is not a good physical volume backing the logical volume). See "Recreating a logical volume on functional disks (UNIX and Windows NT)" on page 139.

AIX logical volumes. You can ensure that a logical volume is backed by a functional AIX operating system logical volume in one of the following ways:

- Move the AIX operating system logical volume onto functional disks (when there is at least one good mirror for the logical volume). See "Moving an AIX logical volume to functional disks (AIX logical volumes)" on page 141.

- Remap the logical volume to a different, functional AIX logical volume (when there is not a good physical volume backing the logical volume). See "Remapping a logical volume to functional AIX volumes (AIX logical volumes)" on page 141.

Moving a physical volume to functional disks (UNIX and Windows NT)

To ensure that a mirror or volume is backed by functional disks, use the **tkadmin move pvol** command to move the physical volume from failed disks to functional disks. This command copies as much of the existing physical volume as possible but does not guarantee that the new physical volume is an accurate copy if there are read errors on the original physical volume.

Perform the following steps to move a physical volume to functional disks:

1. Restart the server in administrative mode. See "Using administrative mode" on page 95 for instructions.
2. Create a new physical disk. On UNIX, use an operating system utility to create a disk or disk partition. On Windows NT, use the Disk Administrator to create a new disk partition or the Encina **fileVol** program to create a new, fully allocated operating system file.
3. Use the **tkadmin init disk** command to initialize the disk. The command syntax follows:

   ```
   tkadmin init disk -server server_name disk_name
   ```

   You must specify the name of the disk to be initialized as the *disk_name* argument.

   On UNIX. For example, enter the following command to initialize a disk named **/dev/rsd0f**:

   ```
   % tkadmin init disk /dev/rsd0f
   ```

   On Windows NT. For example, enter the following command to initialize a disk named **D:\rqs1data**:

   ```
   C:\> tkadmin init disk D:\rqs1data
   ```

4. Use the **tkadmin move pvol** command to move the physical volume. The command syntax follows:

   ```
   tkadmin move pvol -server server_name physical_volume_name
   number_of_regions {{source_disk_name source_disk_offset region_size
   destination_disk_name destination_disk_offset}...}
   ```

   You must specify the name of the physical volume for the *physical_volume_name* argument and the number of regions for the *number_of_regions* argument. For each region, you must specify the source disk name for the *source_disk_name* argument, the source disk offset for the *source_disk_offset* argument, the size of the region for the *region_size* argument, the destination disk name for the *destination_disk_name*

argument, and the destination disk offset for the *destination_disk_offset* argument. The disk space specified for the destination disk must be at least as large as the disk space of the original (source) disk.

You can use Enconsole (or the **tkadmin query lvol** command) to obtain information about the physical volume needed in this command. Moving physical volumes does not affect the mapping of physical volumes to logical volumes. (Deleting and recreating the logical volume is *not* necessary.)

Note: You must move all regions that backed the original physical volume. If an original source disk is not backing a physical volume after this operation, it can be removed.

On UNIX. For example, enter the following command to move a region (with an offset of 0 (zero) and size of 1984) of the physical volume named **pvol8**. The region is moved from disk partition **/dev/rsd1f** to disk partition **/dev/rsd2g** at an offset of **0**:

```
% tkadmin move pvol pvol8 1 /dev/rsd1f 0 1984 /dev/rsd2g 0
```

On Windows NT. For example, enter the following command to move a region (with an offset of 0 (zero) and size of 1984) of the physical volume named **pvol8**. The region is moved from the disk **D:\serverdata** to the disk **F:\serverdata** at an offset of **0**:

```
C:\> tkadmin move pvol pvol8 1 D:\serverdata 0 1984 F:\serverdata 0
```

Recreating a logical volume on functional disks (UNIX and Windows NT)

To ensure that a failed mirror or volume is backed by functional disks, delete and recreate the logical volume on functional disks. This procedure destroys all data in the logical volume. The data must be restored.

Perform the following steps to recreate a logical volume on functional disks:

1. Restart the server in administrative mode. See "Using administrative mode" on page 95 for instructions.
2. Use the **tkadmin delete lvol** command to delete the mapping between the Encina logical volume and the existing physical volumes. The command syntax follows:

   ```
   tkadmin delete lvol -server server_name logical_volume_name
   ```

   You must specify the name of the logical volume for the *logical_volume_name* argument. For example, enter the following command to delete the mapping for a logical volume named **lvol8**:

   ```
   % tkadmin delete lvol lvol8
   ```
3. Create a new physical disk. On UNIX, use an operating system utility to create a disk or disk partition. On Windows NT, use the Disk

Administrator to create a new disk partition or the Encina **fileVol** program to create a new, fully allocated operating system file.

4. Use the **tkadmin init disk** command to initialize the disk. The command syntax follows:

   ```
   tkadmin init disk -server server_name disk_name
   ```

   You must specify the name of the disk to be initialized as the *disk_name* argument.

   On UNIX. For example, enter the following command to initialize a disk named **/dev/rsd0f**:

   ```
   % tkadmin init disk /dev/rsd0f
   ```

   On Windows NT. For example, enter the following command to initialize a disk named **D:\rqs1data**:

   ```
   C:\> tkadmin init disk D:\rqs1data
   ```

5. Use the **tkadmin create pvol** command to create a new physical volume. The command syntax follows:

   ```
   tkadmin create pvol -server server_name physical_volume_name
   chunk_size number_of_regions {{disk_name disk_offset
   [-regionsize region_size]}...}
   ```

   You must specify the name of the physical volume for the *physical_volume_name* argument, the chunk size (in pages) of the physical volume for the *chunk_size* argument, and the number of regions comprising the physical volume for the *number_of_regions* argument. For each region, specify the name of the disk supplying the region for the *disk_name* argument, the offset of the region for the *disk_offset* argument, and optionally, the size of the region for the *region_size* argument.

   The physical volume can contain one or more regions from one or more disks. Further, the physical volume can cover an entire disk partition (when disk offset is 0 (zero) and no region size is specified) or a portion of a disk partition (when the disk offset is nonzero, the region size is specified, or both).

   On UNIX. For example, enter the following command to create a physical volume named **sfspvol2** made up of one region, with a chunk size of **64** and an offset of **0**, on a disk named **/dev/rsd0g**:

   ```
   % tkadmin create pvol sfspvol2 64 1 /dev/rsd0g 0
   ```

   Because the disk offset is 0 (zero) and a region size is not specified, the region encompasses the entire disk partition (partition **/dev/rsd0g**).

On Windows NT. For example, enter the following command to create a physical volume named **sfspvol2** made up of one region, with a chunk size of **64** and an offset of **0**, on a disk named **D:\serverdata**:

```
C:\> tkadmin create pvol sfspvol2 64 1 D:\serverdata 0
```

Because the disk offset is 0 (zero) and a region size is not specified, the region encompasses the entire disk partition (logical drive **D:**).

6. Use the **tkadmin recreate lvol** command to recreate the logical volume. The command syntax follows:

   tkadmin recreate lvol -server *server_name logical_volume_name physical_volume_name*

   This command creates a new mapping from the logical volume to the new functional physical volume. You must specify the name of the logical volume for the *logical_volume_name* argument and the name of the physical volume for the *physical_volume_name* argument. For example, enter the following command to map a logical volume named **lvol8** to a functional physical volume named **pvol8**:

   ```
   % tkadmin recreate lvol lvol8 pvol8
   ```

Moving an AIX logical volume to functional disks (AIX logical volumes)

You must use an AIX administrative utility to move an AIX operating system logical volume. The **tkadmin** commands cannot be used to move AIX logical volumes.

Remapping a logical volume to functional AIX volumes (AIX logical volumes)

If you are using an AIX operating system logical volume to back an Encina logical volume, you must unmap the Encina logical volume and then remap it to a functional AIX logical volume. This procedure destroys all data in the logical volume. The data must be restored.

Perform the following steps to remap a logical volume:

1. Discard the failed AIX logical volume (the space can be reclaimed later).
2. Restart the server in administrative mode. See "Using administrative mode" on page 95 for instructions.
3. Use the **tkadmin unmap lvol** command to unmap the Encina logical volume. The command syntax follows:

   tkadmin unmap lvol -server *server_name logical_volume_name*

   You must specify the name of the Encina logical volume for the *logical_volume_name* argument. For example, enter the following command to unmap a logical volume named **sfslvol1**:

   ```
   % tkadmin unmap lvol sfslvol1
   ```

4. Create a new AIX logical volume (or use an existing one) to replace the failed AIX logical volume. The new AIX logical volume must be at least as large as the original AIX volume.
5. Use the **tkadmin remap lvol** command to remap the Encina logical volume to a functional AIX logical volume. The command syntax follows:

   ```
   tkadmin remap lvol -server server_name logical_volume_name
   operating_system_logical_volume_name
   ```

   You must specify the name of the Encina logical volume for the *logical_volume_name* argument and the name of the AIX logical volume for the *operating_system_logical_volume_name* argument. For example, enter the following command to remap an Encina logical volume named **sfslvol1** to an AIX logical volume named **aix.sfslvol1**:

   ```
   % tkadmin remap lvol sfslvol1 aix.sfslvol1
   ```

Repairing a failed mirror

If a media failure affects one mirror of a mirrored volume, you can return the volume to its normal state by repairing the failed mirror. For Encina logical volumes backed by physical volumes, you synchronize the mirrors. For Encina logical volumes backed by AIX logical volumes, you must use AIX administrative utilities to repair the AIX mirror.

In either case, the server can continue operating. Depending on the needs and use of the specific server, you may choose to shut down a server before repairing the mirror. Restarting the server in administrative mode before repairing a failed mirror prevents further media failure.

On UNIX and Windows NT. Perform the following steps to repair a failed mirror:

1. Make sure that the storage devices backing the failed volume are functional. Refer to the procedures described in "Ensuring that media Is functional" on page 137.
2. If necessary, restart the server in normal operations mode.
3. Synchronize the original copy and the new mirrored copy of the data. (Both copies are referred to as mirrors.) You synchronize the mirrors by using the **tkadmin sync mirrors** command. The command syntax follows:

   ```
   tkadmin sync mirrors -server server_name logical_volume_name
   ```

   You must specify the name of the logical volume for the *logical_volume_name* argument. The command does not return until the synchronization is complete. For example, enter the following command to synchronize all the mirrors backing a logical volume named **sfslvol1**:

   ```
   % tkadmin sync mirrors sfslvol1
   ```

On AIX logical volumes. On the AIX operating system, the mirroring of data can be performed by the operating system or by Encina. To correct the mirroring of an AIX logical volume used to store an Encina logical volume, you must repair the AIX volume with an AIX utility. The **tkadmin** commands cannot be used.

However, if a media failure affecting AIX logical volumes results in a loss of data, the data must be restored using Encina (not AIX utilities). Instructions for restoring volumes are provided in "Restoring a data volume" and "Restoring a log volume" on page 147.

Restoring a data volume

For the restore process to succeed, the server's log volume (log file) must be undamaged (or must already have been restored, if lost). The restore procedure for a data volume has two phases. In the first phase, the server's recovery service component copies the contents of the backup files onto the volume. A backup is identified by the sequence number of the *most recent* file that is part of the backup (the one with the highest number). The oldest backup file is copied first, and the newest backup file is copied last.

All backup files need not be online at the same time. During restoration, if an offline backup file is needed, the restore command returns a "bad backup file number" error message. You can then bring the backup file online and resume the restore by reissuing the restore command. The restore process can be resumed from where it left off; the backup files that have already been used (copied to the new logical volume) are no longer needed online.

For example, consider a backup that contains files with sequence numbers 0 (zero) through 6. Only files 4 through 6 are online at the beginning of the restore. When the restore command returns with a "bad backup file number" error message, you can bring the backup files numbered 0 (zero) through 3 online and resume the restore by reissuing the command.

The second phase of the restore begins after all the backup files have been copied onto the volume. In this phase, all updates to the volume made since the backup are redone using the records in the server's log file (from the oldest data in the archive files to the latest data in the log file). This may require the server to read log data from archive files that are no longer online. In that case, the server issues a message requesting an offline archive file. You must monitor the server during a restore and respond to any requests.

A summary of the restore procedure follows:

1. Restart the server in administrative mode.
2. Remove all mirrors of the logical volume.
3. Ensure that media is functional.

4. Restore data volumes.
5. Monitor the restore process, and, if necessary, move archive files online.
6. Recover the server's data volumes.
7. If you are restoring a cell manager's data volume, resynchronize the Monitor cell repository.
8. Add new mirrors and synchronize them.
9. Restart the server in normal operations mode.

The procedure to restore a server's data volumes from backup files and log archive files is the same for all devices used to back Encina logical volumes.

Perform the following steps to restore a data volume:

1. Restart the server in administrative mode. See "Using administrative mode" on page 95 for instructions.
2. Use the **tkadmin remove mirror** command to remove all mirrors of the logical volume (leaving only one copy of the data to be restored). This command must be issued once for each mirror being removed. The command syntax follows:

   ```
   tkadmin remove mirror -server server_name logical_volume_name
   physical_volume_name
   ```

   You must specify the logical volume name for the *logical_volume_name* argument and the physical volume name of the mirror to be removed for the *physical_volume_name* argument. (You can view information about which physical volumes back a logical volume by using Enconsole or the **tkadmin query lvol** command.)

   For example, enter the following command to remove the mirror named **sfspvol1** backing the logical volume **sfslvol1**:

   ```
   % tkadmin remove mirror sfslvol1 sfspvol1
   ```
3. Make sure that the failed volume is backed by functional disks. Refer to the procedures described in "Ensuring that media Is functional" on page 137.
4. Use the **tkadmin restore lvols** command to restore the data volumes. The command syntax follows:

   ```
   tkadmin restore lvols -server server_name number_of_volumes
   {{logical_volume_name [-backupfilenum backup_file_number]}...}
   [-noresume]
   ```

   You must specify the number of logical volumes to be restored as the *number_of_volumes* argument. For each logical volume, you specify the name of the logical volume for the *logical_volume_name* argument and (optionally) a backup file number for the *backup_file_number* argument. The default backup file number is the number of the last complete backup of

the volume. For example, enter the following command to restore a logical volume named **testVol** using backup file number 5:

```
% tkadmin restore lvols 1 testVol -backupfilenum 5
```

Because each server has a single log file, restoring multiple volumes via separate invocations of the restore command requires reading the same log file several times. To avoid this, the restore command accepts a list of volumes to be restored concurrently. However, you can have only one outstanding restore request at a time. Do not issue a second restore command until the first is complete.

5. Use the **tkadmin query restore** command to monitor the progress of a restore. This command displays the names of the volumes that are being restored, the number of log records that have been replayed, the archive file being processed, and specific information about each volume being restored. The command syntax follows:

   tkadmin query restore -server *server_name*

   For example, enter the following command to display information about the restore process:

   ```
   % tkadmin query restore

   Number of volumes being restored by the server: 1
   So far, 0 log records have been replayed
   Processing MRA file: sfslogvol.LA.13
   Restore of Volume testVol:
    State: copied 2 out of 6 files.
    Backup Number: 5
    Current File:
    <path of backup file>testVol.TRB.000002
    Next File Required:
     <path of backup file>testVol.TRB.000003
   ```

   During the restore process, log archive files that are offline may be needed. You must monitor the warning messages generated by the server during a restore in case archive files need to be moved online. An example message follows:

   ```
   OP MSG: Fetch archive file sfslogvol.LA.14
   After fetching, invoke tkadmin enable archfile.
   ```

   If you receive a warning message requesting an offline archive file, use an operating system utility to move the archive file online. Then, use the **tkadmin enable archfile** command to notify the server that the archive file is available. The command syntax follows:

   tkadmin enable archfile -server *server_name archive_file_name*

You must specify the name of the archive file as the *archive_file_name* argument. For example, the following command enables archive file **sfslogvol.LA.14**:

```
% tkadmin enable archfile sfslogvol.LA.14
```

6. After restoring failed volumes, use the **tkadmin recover lvols** command to recover the server's data volumes. The command brings a server's data volumes up- to-date with the information in the server's log file, if any, and mounts the enabled logical volumes, if any. The command syntax follows:

   tkadmin recover lvols -server *server_name*

   For example, enter the following command to recover data volumes:

   ```
   % tkadmin recover lvols
   ```

7. If you are restoring a cell manager's data volume, resynchronize the attributes in the Monitor cell repository. To do this, use the **enccp** interface to change the value of the *action* attribute of the cell object to **resync** as shown in the following example:

   ```
   enccp> encinaCellManager modify -action resync
   ```

   See "Notes on restoring a cell manager's data volume" for detailed instructions.

8. Add new mirrors and synchronize them.
9. Restart the server in normal operations mode.

Notes on restoring a cell manager's data volume

After you restore a cell manager's data volume, you must resynchronize the values of some special repository attributes maintained by the running node managers and the cell manager. Because a backup of a cell manager's repository captures the state of that cell *at a single point in time*, the repository information does not necessarily reflect the actual state of the nodes and servers in the cell at the time you do the restoration. To bring the repository up-to-date, the cell manager must retrieve current attribute values from the node managers. Therefore, you must use the **enccp** interface to modify the value of the *action* attribute of cell and node objects, forcing a resynchronization of these internal attributes between the cell manager and the running node managers.

To invoke an interactive **enccp** session, type **enccp** and press <**Return**>. The following prompt signals that you have entered interactive mode:

```
enccp>
```

Connect **enccp** to an Encina Monitor cell. The *cell_name* argument must be the fully qualified CDS pathname of the cell. The following example connects **enccp** to the cell named **/.:/trader**:

```
enccp> encinaCellManager connect /.:/trader
```

Resynchronize the attributes in the cell repository by changing the value of the *action* attribute of the cell object to **resync** as shown in the following example. The valid values for the *action* attribute are **none** (the default) and **resync**.

```
enccp> encinaCellManager modify -action resync
```

The cell manager issues audit messages when the resynchronization begins and when it is completed. Concurrent resynchronization requests are not permitted.

You can resynchronize the entire cell or synchronize nodes one at a time as follows:

- Cellwide resynchronization. To resynchronize all nodes in a cell, change the value of the cell object's *action* attribute to **resync**. The cell manager attempts to contact all defined nodes and initiates resynchronization with the nodes it is able to contact. (Setting the action attribute for the cell object is the same as setting the action attribute of each node object in the cell.) The cell manager issues a warning for those nodes that cannot be resynchronized—for example, for nodes that are stopped. If servers are running on a node but the node is not running, change the value of the node object's action attribute after restarting the node.

  Note: Resynchronizing the entire cell blocks access to the repository until the resynchronization is complete—that is, attempts to start new enconsole processes or to perform operations in existing enconsole processes must wait until the cell can be contacted. If you want to maintain access to the cell during resynchronization, it is recommended that you use per-node synchronization.
- Per-node synchronization. You can resynchronize nodes in a particular order (based on importance, activity, or number of servers, for example). First determine which nodes are running and then resynchronize just the running nodes. To resynchronize a node, change the value of the node object's *action* attribute to **resync**. This starts resynchronization of the desired node.

Restoring a log volume

You can restore a server's log volume if you have enabled media archiving and periodically moved archive files to secure offline storage. If you have stored a server's log archive files, a log volume can be restored to a consistent state. However, some log data can be lost. Log data stored in the log file but

not yet moved to archive files is lost, and any online archive files (including the most recent archive file) can be lost. Move archive files offline to safe storage at regular intervals to minimize the potential loss.

Log archive files still online at the time of failure may not be lost. Always attempt to collect the most recent archive files available. Using the last complete archive file available enables you to restore the log volume to the most recent state possible.

The following is a summary of the restore procedure for log volumes:

1. Restart the server in administrative mode.
2. Ensure that media is functional.
3. Restore the log volume.
4. Monitor the restore process.
5. Enable the log file.
6. Restore the server's data volumes (see "Restoring a data volume" on page 143).

The procedure for restoring log volumes from log archive files is the same for all devices used to back Encina logical volumes.

Perform the following steps to restore a log volume:

1. Restart the server in administrative mode. See "Using administrative mode" on page 95 for instructions.
2. Make sure that the log volumes are backed by functional disks. Refer to the procedures described in "Ensuring that media Is functional" on page 137.
3. Use the **tkadmin restore logvol** command to restore the log volume. The command syntax follows:

   ```
   tkadmin restore logvol -server server_name log_volume_name
   archive_device_name archive_file_name
   ```

   You must specify the name of the log volume as the *log_volume_name* argument, the name of the archive device associated with the log volume as the *archive_device_name* argument, and the name of the *latest* archive file as the *archive_file_name* argument.

   The server reads in a range of archive files ending with the specified archive file. It uses these files to restore the log volume's contents. At least the latest archive files to be used in the restore should be available online. As other archive files are required, the server displays a message requesting that the necessary archive file be fetched. After fetching the

required archive file, use the **tkadmin enable archfile** command to inform the server that the file is available. The restore procedure continues, using the requested archive file.

On UNIX. For example, enter the following command to restore a log volume named **sfslogvol** using archive files in the **logarchive** directory. The command uses archive files up to the **sfslogvol.LA.27** archive file (sequence number 27):

```
% tkadmin restore logvol sfslogvol logarchive logarchive/sfslogvol.LA.27
```

On Windows NT. For example, enter the following command to restore a log volume named **sfslogvol** using archive files in the **logarchive** directory. The command uses archive files up to the **sfslogvol.LA.27** archive file (sequence number 27):

```
C:\> tkadmin restore logvol sfslogvol logarchive logarchive\sfslogvol.LA.27
```

4. Monitor the restore process. During the restore process, log archive files that are offline may be needed. You must monitor the warning messages generated by the server during a restore in case archive files need to be moved online. An example message follows:

   ```
   OP MSG: Fetch archive file sfslogvol.LA.14
   After fetching, invoke tkadmin enable archfile.
   ```

   After receiving a warning message asking for an archive file, use an operating system utility to move the archive file online. Then, use the **tkadmin enable archfile** command to notify the server that the archive file is available. The command syntax follows:

   tkadmin enable archfile -server *server_name archive_file_name*

   You must specify the name of the archive file as the *archive_file_name* argument.

   On UNIX. For example, enter the following command to enable an archive file named **sfslogvol.LA.14**:

   ```
   % tkadmin enable archfile logarchive/sfslogvol.LA.14
   ```

   On Windows NT. For example, enter the following command to enable an archive file named **sfslogvol.LA.14**:

   ```
   C:\> tkadmin enable archfile logarchive\sfslogvol.LA.14
   ```

5. Use the **tkadmin enable logfile** command to enable the log file. The command syntax follows:

   tkadmin enable logfile -server *server_name log_file_name*

   You must specify the name of the log file as the *log_file_name* argument. The format is *log_volume*/*log_file_name*, where *log_volume* is the volume on

which the log file is stored—for example, **sfslogvol/sfslogfile**. You can obtain the name of the log file by viewing the Enconsole Recovery Options form or by using the **tkadmin query logvol** command. If you used Enconsole to define and start your server, the default name of the log file is **logFile** and the default name of the log volume is **logVol**. The following command enables a log file named **sfslogfile** on the volume **sfslogvol**:

```
% tkadmin enable logfile sfslogvol/sfslogfile
```

6. Restore all data volumes. This synchronizes the server's data volumes with its log file. This must be done regardless of whether the data volumes were damaged. See "Restoring a data volume" on page 143 for instructions on using backup files and log data to restore a data volume.

   CAUTION:
   After restoring the log file, any attempt to start a server in normal operations mode may corrupt its log file. *Be sure to remain in administrative mode to restore data volumes***. Note that you do not have to stop and restart the server in administrative mode. After you restore and enable the log file, the server is in the same state as it would be if started in administrative mode with a valid log file.**

Part 3. Server security and performance

Chapter 8. Understanding and controlling access to Encina resources

Encina administrators can protect objects against unauthorized access by using Distributed Computing Environment (DCE) access control lists (ACLs). This chapter describes the ACLs and permissions associated with Encina resources. It lists the changes made to the local file system and to DCE when Encina servers are cold started in Enconsole, including changes to the Cell Directory Service (CDS) and to default ACLs. In addition, this chapter describes how to use the **dcecp** interface to modify existing ACLs and to create new ones.

Note: This chapter uses the term "DCE 1.1" to refer to version 1.1 of OSF DCE. Your vendor may use a different version number to refer to the same release of DCE from OSF. See your vendor's DCE documentation for information about the control programs used with your version of DCE.

Overview of ACLs

Access to Encina resources is controlled by ACLs that specify which permissions are granted to which users (principals) and groups. When you use Enconsole to configure a server, the required ACLs for the server are either automatically inherited or created to grant permissions to the appropriate principals and groups. The ACLs for each type of Encina resource—the cell manager, node manager, Monitor application server (MAS) and its interfaces and functions, Recoverable Queueing Service (RQS) server and its associated queues and queue sets, Structured File Server (SFS) server and its associated files, and Peer-to-Peer Communications (PPC) Gateway server—are discussed in detail later in this chapter.

When an Encina server is cold started, the principal named as the exclusive authority has sole access to the server. This principal must grant permissions to objects in the server and then clear the exclusive authority so that permissions are checked. If you are using Enconsole, the default exclusive authority principal is **encina_admin** (a member of the group **encina_admin_group**). Enconsole automatically grants full administrative permissions to members of the group **encina_admin_group**, and then clears the exclusive authority. As necessary, you can grant additional permissions (for example, granting permissions to a client application so that it can access objects in a server).

In addition to creating default ACLs and ACL entries, cold starting an Encina server creates the following:

- CDS entries
- Security objects (such as principals, groups, and accounts), which are added to the DCE Security Registry
- Local keytab files

These CDS entries, security objects, and keytab files are all used in conjunction with ACLs to control access to Encina resources.

Types of ACLs

This chapter covers three types of ACLs:

- Object ACL—An ACL that controls access to an object, such as a directory, a server, or an interface.
- Initial container ACL—An inherited ACL that specifies the ACL entries initially associated with all subdirectories created within a directory.
- Initial object ACL—An inherited ACL that specifies the ACL entries initially associated with all nondirectory objects created within a directory.

Each DCE object has an object ACL. Directory objects (for example, CDS directory entries) have an object ACL, an initial container ACL, and an initial object ACL. Note that Encina resources do *not* have an initial container ACL. Encina servers can have only an object ACL, and RQS queues and queue sets can have both an object ACL and an initial object ACL.

It is important to note that there are two identically named object ACLs for each Encina server. The first ACL controls access to the server itself. The second ACL controls access to that server's entry in CDS.

ACL entry format

Each ACL contains one or more ACL entries. ACL entries have the following format:

{*type* [*key*] *permissions***}**

The *type* argument identifies the type of the ACL entry. The following ACL entry types are supported in Encina:

user Specifies the permissions granted to a specific principal.

group Specifies the permissions granted to members of a group.

foreign_user
Specifies the permissions granted to an authenticated principal from a foreign cell.

foreign_group
Specifies the permissions granted to authenticated members of a foreign group.

foreign_other
Specifies the permissions granted to authenticated principals from a foreign cell who are not named by an ACL entry of type **foreign_user**, or who are not members of a group named by an ACL entry of type **foreign_group**.

unauthenticated
Specifies the permissions that can be granted to unauthenticated principals. A principal who requests access via an unauthenticated RPC is granted the permissions for the **unauthenticated** entry. An ACL that does not contain the **unauthenticated** entry denies access to unauthenticated users.

Note: Encina does not support the **any_other** type. Note, however, that CDS *does* support **any_other** (it is used to control access to a server's CDS entry). The **any_other** type specifies the permissions that are granted to authenticated principals who are not otherwise covered by any other type of ACL entry.

The optional *key* argument identifies the principal or group to which an entry applies (for example, the name of a specific principal, such as **encina_admin**, or the name of a specific group, such as **encina_admin_group**); some entry types apply to predefined classifications of principals and so do not accept a key.

The *permissions* argument lists the permissions granted by the entry. Because there are two identically named ACLs for each server, there are also two types of permissions: permissions for directories in the CDS namespace (used for those ACLs that govern access to a server's CDS entry) and permissions for Encina resources (used for those ACLs that govern access to the server itself). Table 6 lists the permissions for CDS directories.

Table 6. ACL permissions for CDS directories

Permissions	Usage
read (**r**)	Allows principals to read the attributes of a CDS entry.
write (**w**)	Allows principals to update the modifiable attributes of a CDS entry (note that the entry's ACL cannot be modified).
delete (**d**)	Allows principals to delete a CDS entry.

Table 6. ACL permissions for CDS directories (continued)

Permissions	Usage
test (**t**)	Allows principals to test whether an attribute of a CDS entry has a particular value without being able to actually view the values.
control (**c**)	Allows principals to change the ACL for a CDS entry.
insert (**i**)	Allows principals to create new CDS entries.
administer (**a**)	Allows principals to administer the replication of CDS directories.

Note: The permissions for Encina resources are described later in this chapter.

For example, the following ACL entry grants the principal **encina_admin** the **read** (**r**), **delete** (**d**), **test** (**t**), and **insert** (**i**) permissions for a CDS directory (each dash (**-**) in the entry indicates a permission that is not granted):

```
{user encina_admin r-dt-i-}
```

The following ACL entry grants members of the group **encina_admin_group** the **read** (**r**), **write** (**w**), and **delete** (**d**) permissions for a CDS directory:

```
{group encina_admin_group rwd----}
```

ACL evaluation

Access to a directory in the CDS namespace is controlled by the object ACL of the directory. When a principal tries to access a directory, CDS first determines whether the principal has a **user** entry on the directory's ACL. If a matching **user** entry exists, CDS grants access based on the permissions of the **user** entry. If a principal does not have a **user** entry, CDS determines whether the principal belongs to any groups that have **group** entries on the ACL. If one or more matching **group** entries exist, CDS grants access based on the permissions of the **group** entries. If the principal does not have a **user** entry and does not belong to any groups that have a **group** entry on the ACL, CDS continues to evaluate the ACL, checking for a **foreign_user**, **foreign_group**, **foreign_other**, **any_other**, or **unauthenticated** entry, until a matching entry is found.

If the principal matches no entries on the ACL, CDS denies the principal access to the directory. You can also deny a user access by creating an ACL entry for the user with no permissions. The following ACL entry denies access to the principal **fred**:

```
{user fred -------}
```

Authorization to use resources managed by specific Encina servers is granted the same way as for CDS directories. For example, when a PPC Executive application tries to access a gateway server, the gateway server examines its ACL to determine whether the principal for the application has the appropriate permissions. Suppose the **encina_ppcexec_app1** principal attempts to schedule a conversation through the gateway server. The server checks for a **user** entry for **encina_ppcexec_app1** and, finding that ACL entry, checks for the **execute** (**x**) permission on the entry. The **encina_ppcexec_app1** principal is granted **execute** (**x**) permission, so the request is accepted. The ACL entry has the following format:

```
{user encina_ppcexec_app1 -x}
```

Now suppose that the principal **user1**, who belongs to the group **encina_admin_group**, attempts to modify the gateway server's configuration. The gateway server first checks for a **user** entry for **user1**; finding no **user** entry for **user1**, the gateway server then determines whether **user1** belongs to any groups that have entries on the ACL. The principal **user1** belongs to the group **encina_admin_group**, which has **administer** (**a**) permission, so **user1** can query and modify the configuration. The ACL entry has the following format:

```
{group encina_admin_group ax}
```

Using the dcecp interface to work with ACLs

The DCE 1.1 interface used to modify ACLs is the **dcecp** control program. (On IBM DCE for Windows NT, the Visual DCE ACL Editor can also be used. See your IBM DCE documentation for more information on using the Visual DCE ACL Editor.)

The **dcecp** control program has two modes: command-line mode and interactive mode. You can use either mode for most operations.

Note: The **dcecp** control program often requires you to surround arguments with braces ({ }). In command-line mode, many command shells require you to use single quotes (' ') or backslashes (\ \) to escape the braces; see "Adding entries to an ACL" on page 159 for an example. Be careful not to confuse the braces used with **dcecp** commands with the braces used to indicate a list of options in a syntax description; the former are shown in bold font and the latter are not.

Do *not* use the single quotes or backslashes in interactive mode.

- *Command-line mode*. To use command-line mode, enter the name of the control program, **dcecp**, followed by the **-c** option at the system prompt. In command-line mode, each **dcecp** command has the following format:

  ```
  % dcecp -c object operation
  ```

The *object* argument is the type of DCE entity (for example, **acl** or **account**) on which you want to perform an operation, and the *operation* argument is the action (for example, **list** or **show**) that you want to perform on the specified object. Most commands require additional arguments and options to indicate specific instances of objects on which operations are to be performed and the attributes to be associated with those objects and operations.

- *Interactive mode*. To use interactive mode, enter **dcecp** at the system prompt; the `dcecp>` prompt replaces the system prompt. In interactive mode, each **dcecp** command has the following format:

  ```
  dcecp> object operation
  ```

  The *object* and *operation* arguments have the same meaning in both interactive and command-line modes. As in command-line mode, most commands require additional arguments and options to provide specific information about objects and operations.

  To exit from interactive mode, enter **exit** at the `dcecp>` prompt.

Remember that some resources have more than one ACL (see "Types of ACLs" on page 154). To enable you to specify the ACL you want to manipulate, the **dcecp** command has the following options:

- The **-e** option, which directs a command to manipulate the ACL for a CDS entry
- The **-ic** option, which directs a command to manipulate the initial container ACL of a directory
- The **-io** option, which directs a command to manipulate the initial object ACL of a directory

If you want to manipulate the ACL for an Encina resource itself, do *not* use any of these options with the **dcecp** command.

Listing the entries on an ACL

To list the entries on an ACL, use the **dcecp acl show** command as follows, where *path_name* is the pathname of the ACL:

```
% dcecp -c acl show path_name [{-e | -ic | -io}]
```

In the example below, the **dcecp acl show** command displays the entries on the ACL that controls access to an RQS server named **rqs1**.

```
% dcecp -c acl show /.:/branch1/server/rqs1
{group encina_admin_group caxtq}
{group encina_operator_group q}
```

In the next example, the **dcecp acl show** command displays the entries on the ACL that controls access to the CDS entry for the RQS server called **rqs1**. The

inclusion of the **-e** option means that the entries displayed are for the RQS server's CDS entry, not for the RQS server itself.

```
% dcecp -c acl show /.:/branch1/server/rqs1 -e
{unauthenticated r--t---}
{user branch1/server/rqs1 rwdtc--}
{group subsys/dce/cds-admin rwdtc--}
{group subsys/dce/cds-server rwdtc--}
{group encina_admin_group rwdtc--}
{group encina_servers_group rwdtc--}
{any_other r--t---}
```

Saving an ACL to a file

You can direct the output of the **dcecp acl show** command to a file by using the following command, where *path_name* is the pathname of the ACL, and *output_file* is the name of the file in which you want to save the output of the command:

```
% dcecp -c acl show path_name [{-e | -ic | -io}] > output_file
```

If the ACL ever becomes damaged, you can use the information stored in the output file to restore its entries. In addition, you can use the contents of the output file to copy that ACL's entries to another ACL.

Adding entries to an ACL

To add an entry to an ACL, use the **dcecp acl modify** command with the **-add** option as follows, where *path_name* is the pathname of the ACL, and *ACL_entry* is the new entry you want to add:

```
% dcecp -c acl modify path_name [{-e | -ic  | -io}] -add '{ACL_entry}'
```

To add multiple entries, use a slightly different syntax with the command, as follows:

```
% dcecp -c acl modify path_name [{-e | -ic  | -io}]  \
-add '{{ACL_entry} {ACL_entry}...}'
```

Note: To make ACL management easier, it is recommended that you avoid adding individual user entries to an ACL whenever possible. Instead, use group entries; put users into groups (such as the group **encina_admin_group** or **encina_operator_group**), and then add the appropriate groups to the appropriate ACLs.

The following scenario illustrates the benefit of group entries. A user named **user1** needs permissions to manage RQS queues. Instead of adding an individual user entry ({`user user1 ladoxnemrpq`}) to the ACL for each queue, you add **user1** to a group (for example,

encina_admin_group) that has all of the permissions **user1** needs. You then add the group entry ({`group encina_admin_group ladoxnemrpq`} to the ACL for each queue.

You now decide to give another user, named **user2**, permissions to manage the same RQS queues. Because you already have a group that has all necessary permissions and because you have already added the group entry to the ACL for each RQS queue, all you must do is add **user2** to that group. Now both **user1** and **user2** can modify the RQS queues.

Modifying entries on an ACL

To modify an existing ACL entry, use the **dcecp acl modify** command with the **-change** option as follows, where *path_name* is the pathname of the ACL, and *ACL_entry* specifies the new entry, including the new permissions to be associated with the entry:

```
% dcecp -c acl modify path_name [{-e | -ic | -io}] -change '{ACL_entry}'
```

To modify multiple entries, use a slightly different syntax with the command, as follows:

```
% dcecp -c acl modify path_name [{-e | -ic | -io}] \
-change '{{ACL_entry} {ACL_entry}...}'
```

Controlling access to a Monitor cell

This section describes the Monitor cell administrative ACL and the permissions associated with it. It also describes the changes made to the file system and to DCE when a Monitor cell is cold started.

The Monitor cell administrative ACL

Each Monitor cell that is started with DCE security has an ACL that controls administrative access to the cell. The cell's administrative ACL is named **/.:/***cell_name***/ecm**, where *cell_name* is the name of the cell.

Table 7 lists the possible administrative permissions for Monitor cells.

Table 7. Administrative ACL permissions for Monitor cells

Permissions	Usage
administer (**a**)	Allows principals to create and modify ACLs for Monitor administration and application server interfaces.

Table 7. Administrative ACL permissions for Monitor cells (continued)

Permissions	Usage
operator (**o**)	Allows principals to start and stop all application servers and issue **tkadmin** commands on Monitor application servers without allowing the principals to create or modify objects in the repository.
read (**r**)	Allows principals to list and query information in the Monitor repository.
write (**w**)	Allows principals to create and modify information in the Monitor repository.

When a Monitor cell is first started, the principal named as the exclusive authority is granted exclusive access to the cell manager. If you are using Enconsole, the default exclusive authority is the user **encina_admin**. Enconsole automatically grants full administrative permissions to the **encina_admin_group** and then clears the exclusive authority.

If you are not using Enconsole, the exclusive authority must create entries on the cell manager ACL. For example, the **read** (**r**), **write** (**w**), and **administer** (**a**) permissions for the Monitor cell can be granted by adding the appropriate entry to the cell manager ACL. See "Modifying the Monitor cell ACL" on page 163 for more information on modifying the cell manager ACL. After permissions have been set, the exclusive authority must be cleared by using the **tkadmin clear exclusiveauthority** command.

Effects of starting a Monitor cell

This section describes the changes that are made to the file system and to DCE when a cell manager is started for the first time.

Effects on the file system

When a Monitor cell is cold started, the following directories are created:

- The **/opt/encinalocal/***cell_name***/ecm** directory—the working directory for the cell manager. By default, it contains the following files: **ecm.log**, **keyfile**, **rc.encina.cell** (**encina.cell.cmd** on Windows machines), **restart**, and **server.out**. It also contains the **/logArchive** directory.
- The **/opt/encinamirror/***cell_name***/ecm** directory—the mirror directory for the cell manager. By default, it contains the **restart.bak** file.

Note: On HP-UX machines, the working directory is **/var/opt/encina/local/***cell_name***/ecm**, and the mirror directory is **/var/opt/encina/mirror/***cell_name***/ecm**.

Effects on DCE

When a Monitor cell is cold started, several changes are made to the DCE Security Registry and to the CDS namespace. In addition, default ACLs and ACL entries are created. The following sections describe the changes in detail.

Changes to the Security Registry: When a Monitor cell is cold started, the following changes are made to the Security Registry:

- The following principals are created:
  - **encina_admin**
  - **encina_operator**
  - *cell_name*/**ecm**
  - *cell_name*/**server/_init**
  - *cell_name*/**node/_init**
- Accounts are created for the **encina_admin**, **encina_operator**, and *cell_name*/**ecm** principals.
- The following groups are created:
  - **encina_admin_group**
  - **encina_operator_group**
  - **encina_servers_group**
- The **encina_admin** principal is added to the groups **encina_admin_group** and **encina_operator_group**. The **encina_operator** principal is added to the group **encina_operator_group**.

Once these security objects are created, the cell manager's keytab file (by default, **keyfile**) is updated to include the cell manager principal and password.

Changes to CDS: When a Monitor cell is cold started, the following changes are made to CDS:

- The CDS path **/.:/***cell_name* is created. This is the DCE namespace for the cell itself. In addition, the following directories are created in the cell's namespace:
  - **/.:/***cell_name*/**node**
  - **/.:/***cell_name*/**server**
  - **/.:/***cell_name*/**objects**
  - **/.:/***cell_name*/**trpc**
  - **/.:/***cell_name*/**ppc**
  - **/.:/***cell_name*/**ppc/gateway**
- The CDS entry for the cell manager (**/.:/***cell_name*/**ecm**) is created.

Note: This CDS entry stores the cell manager's binding information. Included in the binding information is a Name Service Interface (NSI) server entry that serves as the cell manager's junction. (A junction is a branch into the non-CDS part of the DCE namespace. It enables connections to servers outside of CDS.)

Changes to ACLs: Default ACLs are created for each DCE object associated with the Monitor cell, including directories and the cell manager itself. Once the ACLs are created, default entries are added to them.

- An object ACL, initial container ACL, and initial object ACL—all called **/.:/***cell_name*—are created. These are the ACLs for the cell's namespace. In addition, object ACLs, initial container ACLs, and initial object ACLs are created for all of the directories contained within the cell's namespace (refer to "Changes to CDS" on page 162 for a complete list of directories). The following entries are added to all of the new ACLs:
  - `{group encina_admin_group rwdtc--}`
  - `{group encina_servers_group rwdtc--}`
- Two **/.:/***cell_name***/ecm** ACLs are created for the cell manager. The first ACL controls access to the cell manager itself. The following entries are added to this ACL:
  - `{group encina_admin_group rwao}`
  - `{group encina_operator_group r--o}`

  The second ACL controls access to the CDS entry for the cell manager. The following entries are added to this ACL:
  - `{unauthenticated r--t---}`
  - `{user` *cell_name*`/ecm rwdtc--}`
  - `{group subsys/dce/cds-admin rwdtc--}`
  - `{group subsys/dce/cds-server rwdtc--}`
  - `{group encina_admin_group rwdtc--}`
  - `{group encina_servers_group rwdtc--}`
  - `{any_other r--t---}`

Modifying the Monitor cell ACL

Before modifying the Monitor cell ACL, you must determine the types of tasks users are likely to perform and then assign permissions accordingly. For example, Encina Monitor system administrators must be given Monitor **read** (**r**) permission if they are to query configuration entries and either the **write** (**w**) or the **operator** (**o**) permission, depending on their administrative duties. Because users with the **administer** (**a**) permission on the cell also have the ability to grant themselves other permissions on the cell and its servers, grant the **administer** (**a**) permission cautiously.

The **dcecp acl modify** command enables you to modify the Monitor cell ACL. You must have **administer** (**a**) permission on the Monitor cell to modify the ACL.

The following command adds an entry to the Monitor cell ACL, granting the user **user1** the **read** (**r**) permission for the cell:

```
% dcecp -c acl modify /.:/branch1/ecm -add '{user user1 r}'
```

Controlling access to a node manager

By default, access to a node manager is controlled by the Monitor cell ACL. This section describes the changes made to the file system and to DCE when a node manager is cold started.

In addition, this section describes the node manager administrative ACL that you can create to override the permissions granted by the Monitor cell ACL.

Effects of starting a node manager

When a node manager is cold started, changes are made to both the file system and DCE. This section describes the changes in detail.

Effects on the file system

The following directories are created when a node manager is cold started:

- The **/opt/encinalocal/***cell_name***/node/***node_name* directory—the working directory for the node manager. It contains the following files: **rc.encina.***node_name* (**encina.***node_name***.cmd** on Windows machines), **keyfile**, **restart**, and **server.out**. It also contains the **/logArchive** directory.
- The **/opt/encinamirror/***cell_name***/node/***node_name* directory—the mirror directory for the node manager. It contains the **restart.bak** file.

Note: On HP-UX machines, the working directory is **/var/opt/encina/local/***cell_name***/node/***node_name*, and the mirror directory is **/var/opt/encina/mirror/***cell_name***/node/***node_name*.

Effects on DCE

When a node manager is cold started, several changes are made to the DCE Security Registry and to the CDS namespace. In addition, default ACLs and ACL entries are created.

Changes to the Security Registry: When a node manager is cold started, the following changes are made to the Security Registry:

- The node manager principal (*cell_name***/node/***node_name*) is created.
- An account is created for the node manager principal.

- The node manager principal is added to the groups **encina_admin_group** and **encina_servers_group**.

When the node manager principal and its account have been created, the keytab file is modified to include the node manager principal and password.

Changes to CDS: When a node manager is cold started, the CDS entry **/.:/***cell_name***/node/***node_name* is created.

Note: This CDS entry stores the node manager's binding information. Included in the binding information is a Name Service Interface (NSI) server entry that serves as the node manager's junction to non-CDS namespace.

Changes to ACLs: When a node manager is cold started, the *cell_name***/node/***node_name* ACL is created to control access to the CDS entry for the node manager. The following entries are added to this ACL:

- `{unauthenticated r--t---}`
- `{user` *cell_name*`/node/`*cell_name*` rwdtc--}`
- `{group subsys/dce/cds-admin rwdtc--}`
- `{group subsys/dce/cds-server rwdtc--}`
- `{group encina_servers_group rwdtc--}`
- `{group encina_admin_group rwdtc--}`
- `{any_other r--t---}`

The node manager administrative ACL

Although the Monitor cell ACL is used by default to control access to a node manager, you can create a separate node manager administrative ACL after the node manager has been started. This ACL controls access *only* to the node manager, and it overrides any permissions set by the Monitor cell ACL.

The node manager administrative ACL is named **/.:/***cell_name***/ecm/node/***node_name*, where *cell_name* is the name of the Monitor cell, and *node_name* is the name of the node—for example, **/.:/branch1/ecm/node/machine1**. It is used to grant permissions to principals to perform administrative tasks on the node (for example, creating log files).

Table 8 lists possible administrative permissions for Monitor node managers.

Table 8. Administrative ACL permissions for Monitor node managers

Permission	Usage
administer (**a**)	Allows principals to administer the node.
query (**q**)	Allows principals to query the state of the node.

Creating and modifying a node manager ACL

Before creating the node manager ACL, ensure that you have the Encina Monitor **administer** (**a**) permission, and that the cell manager is running with authorization enabled.

To create an ACL for a node, use the **dcecp acl modify** command. The following example command grants members of the group **encina_admin_group** the **administer** (**a**) and **query** (**q**) permissions on the node **machine1**:

```
% dcecp -c acl modify /.:/branch1/ecm/node/machine1 \
-add '{group encina_admin_group aq}'
```

Once you have created the node manager ACL, you can use the **dcecp acl modify** command to modify the existing ACL entry or to add other entries. Refer to "Using the dcecp interface to work with ACLs" on page 157 for more information on modifying or adding ACL entries.

Controlling access to a Monitor application server

By default, administrative access to a Monitor application server is controlled by the Monitor cell ACL. However, client access to interfaces exported by the application server is controlled by separate interface ACLs.

This section describes the changes made to the file system and to DCE when an application server is cold started. In addition, this section describes the administrative application server ACL and the function ACLs that you can create to override the permissions granted by the Monitor cell ACL and the interface ACLs, respectively.

Effects of starting a Monitor application server

When an application server is cold started, changes are made to the file system and to DCE. This section describes the changes in detail.

Effects on the file system

The following directories are created when an application server is cold started:

- The **/opt/encinalocal/***cell_name***/server/***server_name* directory—the working directory for the server. It contains the keytab file (called **keyfile**).
- The **/opt/encinalocal/***cell_name***/server/***server_name***/pa***pa_number* directory—the processing agent (PA) working directory; it contains the **server.out** file for that PA. Each PA associated with the MAS has its own working directory within the server's working directory.

Note: On HP-UX machines, the server's working directory is **/var/opt/encina/local/***cell_name***/server/***server_name*. The PA's working

directory is
/var/opt/encina/local/*cell_name***/server/***server_name***/pa***pa_number*.

Effects on DCE

When an application server is cold started, several changes are made to the DCE Security Registry and the CDS namespace. In addition, several ACLs are created and default ACL entries are added to them.

Changes to the Security Registry: When an application server is cold started, the following changes are made to the Security Registry:

- The application server principal (*cell_name***/server/***server_name*) is created.
- An account is created for the application server principal.
- The application server principal is added to the group **encina_servers_group**.

Once the application server principal and its account have been created, the keytab file is modified to include the new principal and its password.

Changes to CDS: When an application server is cold started, the following CDS entries are created:

- **/.:/***cell_name***/server/***server_name*
- **/.:/***cell_name***/server/***server_name*_**group**_
- **/.:/***cell_name***/server/***server_name*_**pa***number*

Note: Each PA associated with the application server has its own CDS entry. The following example shows the CDS entry for **pa2**, which belongs to the **billing** server in the **branch1** Monitor cell:

```
/.:/branch1/server/billing_pa2
```

If you have an application server that uses the Encina Object Transaction Service (OTS), the following additional CDS entry is created:

- **/.:/***cell_name***/objects/***uuid***:***interface_name* (where *uuid* is the universal unique identifier for the interface specified by *interface_name*)

Changes to ACLs: When an application server is cold started, the following changes are made:

- An ACL is created for the interface you selected while defining the server; it is named **/.:/***cell_name***/ecm/interface/***interface_name*. The following entry is added to the interface ACL:

  ```
  {group encina_admin_group x}
  ```

  (For more information on interface ACLs, see "Controlling access to interfaces and functions" on page 169.)

- The **/.:/***cell_name***/server/***server_name* ACL is created to control access to the application server's primary CDS entry. The following entries are added to this ACLs:
  - {unauthenticated r--t---}
  - {user *cell_name*/server/*server_name* rwdtc--}
  - {group subsys/dce/cds-admin rwdtc--}
  - {group subsys/dce/cds-server rwdtc--}
  - {group encina_admin_group rwdtc--}
  - {group encina_servers_group rwdtc--}
  - {any_other r--t---}

The Monitor application server administrative ACL

Although the Monitor cell ACL is used by default to control administrative access to an application server, you can create a separate administrative ACL after the application server has been started. This application server ACL controls access only to the server, and it overrides any permissions granted by the Monitor cell ACL. It is used to grant permissions to principals to perform administrative tasks such as listing and aborting transactions.

The application server ACL is named **/.:/***cell_name***/ecm/server/***server_name*, where *cell_name* is the name of your Monitor cell, and *server_name* is the name of the application server—for example, **/.:/branch1/ecm/server/merchandise**.

Table 9 lists possible administrative permissions for application servers.

Table 9. Administrative ACL permissions for application servers

Permission	Usage
administer (**a**)	Allows principals to administer the server.
query (**q**)	Allows principals to query the state of the server.

Creating and modifying an application server ACL

Before creating or modifying an application server ACL, ensure that you have the Encina Monitor **administer** (**a**) permission and that the cell manager is running with authorization enabled.

To create an ACL for an application server, use the **dcecp acl modify** command. The following example command grants members of the group **encina_admin_group** the **administer** (**a**) and **query** (**q**) permissions on the server **merchandise**:

```
% dcecp -c acl modify /.:/branch1/ecm/server/merchandise \
-add '{group encina_admin_group aq}'
```

After you have created the application server ACL, you can use the **dcecp acl modify** command to modify the existing ACL entry or to add other entries. Refer to "Using the dcecp interface to work with ACLs" on page 157 for more information on modifying or adding ACL entries.

Controlling access to interfaces and functions

When an application server is cold started, an interface ACL is created for all defined interfaces; it controls client access to the exported interface and its functions. Interface ACLs are named **/.:/***cell_name***/ecm/interface/***interface_name*, where *interface_name* is the name of the interface exported by the Monitor application server (for example, **/.:/branch1/ecm/interface/merchandise**).

By default, an interface ACL controls access to the functions associated with the interface. If you want to override the permissions granted by an interface ACL for any function in that interface, you can create separate function ACLs. Function ACLs are named **/.:/***cell_name***/ecm/interface/***interface_name***/***function_name*, where *function_name* is the name of the function (for example, **/.:/branch1/ecm/interface/merchandise/write_checks**). Each function ACL controls access to one specific function.

Table 10 lists possible client permissions for interfaces and functions.

Table 10. Client ACL permissions for interfaces and functions

Permission	Usage
execute (**x**)	Allows principals (clients) to use an application interface or function.

Creating and modifying client ACLs for an application interface or function

To create an ACL for a function, you must have the Encina Monitor **administer** (**a**) permission for the function, and the cell manager must be running with authorization enabled. Use the **dcecp acl modify** command, as follows:

```
% dcecp -c acl modify /.:/cell_name/ecm/interface/interface_name/function_name \
-add '{ACL_entry}'
```

For example, the following command grants members of the group **bank_managers** the **execute** (**x**) permission on the ACL for the **write_checks** function:

```
%  dcecp -c acl modify /.:/branch1/ecm/interface/merchandise/write_checks \
-add '{group bank_managers x}'
```

Note: By giving only members of the group **bank_managers** the **execute** (**x**) permission for a specific function, users who do not belong to this group are denied access to the function even if they have permission on the **merchandise** interface.

You can also use the **dcecp acl modify** command to change the entries on either an interface or function ACL. For example, the following command grants members of the group **accounting** the **execute** (**x**) permission on the ACL for the **merchandise** interface:

```
% dcecp -c acl modify /.:/branch1/ecm/interface/merchandise \
-add '{group accounting x}'
```

Controlling access to RQS objects

This section describes the ACLs used to control access to an RQS server and its queues and queue sets. It also describes the changes made to the file system and to DCE when an RQS server is cold started.

The RQS server, queue, and queue set administrative ACLs

Each RQS server, queue, and queue set has its own ACL. The following sections describe these ACLs and the permissions associated with them.

The RQS server ACL

Each RQS server has an ACL that governs access to it. The RQS server ACL is called **/.:/***cell_name***/server/***server_name*, where *cell_name* is the name of your Monitor cell and *server_name* is the name of your RQS server. Table 11 lists the permissions possible for an RQS server and its objects.

Table 11. ACL permissions for RQS objects

Permissions	For a Server	For a Queue	For a Queue Set
administer (a)	View or modify server ACL and all other ACLs; administer the server by using **tkadmin** and **rqsadmin** commands. Users with **administer (a)** permission are automatically granted **administer (a)** permission on queues and queue sets.	View or modify queue ACL.	View or modify queue set ACL.

Table 11. ACL permissions for RQS objects (continued)

Permissions	For a Server	For a Queue	For a Queue Set
callback (l)	—	Associate a callback with a queue.	—
create (c)	Create queues, queue sets, and element types.	—	—
dequeue (d)	—	Dequeue or delete an element.	Dequeue an element from a queue set.
dequeue orphan (o)	—	Dequeue or delete an element as an orphan.	Dequeue or delete an element from a queue set as an orphan.
destroy (x)	Destroy queues, queue sets, and element types.	Destroy a queue.	Destroy a queue set.
enqueue (n)	—	Enqueue or requeue elements to a queue.	—
exclusive access (e)	—	Acquire or release exclusive access to a queue.	—
modify (m)	—	Modify element data; obtain write locks on elements through ID, key, or cursor access.	Insert or remove queues from a set, assign a queue to a different priority class in a set, modify service levels, and remove priority classes from a set.
query (q)	Display information about a server.	Display information about a queue.	Display information about a queue set.
read element (r)	—	Read element contents; obtain read and upgrade locks on elements through ID, key, or cursor access.	—
reset (p)	—	Reset statistics collection period for a queue.	—

Table 11. ACL permissions for RQS objects (continued)

Permissions	For a Server	For a Queue	For a Queue Set
tune (t)	Manipulate server timeouts, callback retention periods, and table sizes.	—	—

When an RQS server is first started, the principal named as the exclusive authority is granted exclusive access to the server. If you are using Enconsole, the default exclusive authority is the principal **encina_admin**. Enconsole automatically grants full administrative permissions to the group **encina_admin_group** and then clears the exclusive authority.

If you are not using Enconsole, the exclusive authority must create entries on the server ACL. For example, the **create** (**c**), **administer** (**a**), **destroy** (**x**), **tune** (**t**), and **query** (**q**) permissions on the server can be granted by adding an appropriate entry to the server ACL. After permissions have been set, the exclusive authority must be cleared by using the **tkadmin clear exclusiveauthority** command.

The following ACL entry, added to the server ACL by the principal **encina_admin**, grants all server permissions to the group **encina_admin_group**:

```
{group encina_admin_group caxtq}
```

After the exclusive authority is cleared, any user holding the **administer** (**a**) permission on the server can do the following:

- Read, modify, or otherwise access the server ACL itself—for example, add permissions to the server ACL for other users or groups.
- Read, modify, or otherwise access the ACL for any queue or queue set in the server. The holder of the **administer** (**a**) permission on the server is automatically granted **administer** (**a**) permission on all objects in the server.
- Administer the server by using **rqsadmin** and **tkadmin** commands.

The RQS queue and queue set ACLs

Each queue and queue set in an RQS server has an ACL that governs access to it. The queue ACL is called **/.:/***cell_name***/server/***server_name***/queue/***queue_name*. The queue set ACL is called **/.:/***cell_name***/server/***server_name***/qset/***queue_set_name*. Table 11 on page 170 lists the RQS permissions for queue and queue set objects.

To carry out a given operation on a queue or queue set, a user must possess appropriate permissions on the object. For example, to remove a queue from a

queue set, a user must have the **modify** (**m**) permission on the queue set. To destroy a queue, a user must have the **destroy** (**x**) permission on the queue.

A user holding the **create** (**c**) permission on a server can create queues, queue sets, and element types in the server. Any user who creates a queue or queue set is automatically granted full permissions on the queue or queue set. For instance, when a queue is created, its ACL initially consists of an entry that grants the creator all permissions on the queue.

For example, the ACL for a queue created by the principal **encina_admin** automatically contains an entry for that principal. The entry grants the principal all permissions on the queue. The entry has the following format:

```
{user encina_admin ladoxnemrpq}
```

The principal **encina_admin** can then add entries to the ACL for the queue, as follows:

```
{user encina_admin ladoxnemrpq}
{user client1 l-do---mr-q}
{user client2 -----n-----}
{user client3 -----n-----}
{group encina_admin_group -a---------}
```

In this example, the new ACL grants the following permissions:

- The principal **client1** is granted the **callback** (**l**), **dequeue** (**d**), **dequeue orphan** (**o**), **modify** (**m**), **read element** (**r**), and **query** (**q**) permissions on the queue.
- The principals **client2** and **client3** are granted the **enqueue** (**n**) permission on the queue.
- All users belonging to the group **encina_admin_group** are granted the **administer** (**a**) permission on the queue.

To dequeue an element (or an orphan element) from a queue set, a user must have both of the following:

- The **dequeue** (**d**) or **dequeue orphan** (**o**) permission on the queue set
- The **dequeue** (**d**) or **dequeue orphan** (**o**) permission on all queues belonging to the queue set

When a queue is inserted into a queue set, be sure that appropriate permissions are granted to any users who require access to the queue. If one of the permissions is missing, the dequeue operation fails.

When granting permissions to users, be aware that some operations (such as dequeue) implicitly allow other operations (such as read). A user is not prevented from carrying out the implicit operation even if that user does not

explicitly have the corresponding permission. The following are examples of permissions that allow users to carry out other operations for which they do not necessarily hold permissions:

- The **dequeue** (**d**) permission on a queue allows its holder to dequeue (and therefore read) an element, the same operation allowed with the **read element** (**r**) permission.
- The **dequeue orphan** (**o**) permission on a queue allows its holder to dequeue an orphan element (and in doing so, read and modify the element), the same operations allowed with the **read element** (**r**) and **modify** (**m**) permissions.

Effects of starting an RQS server

When an RQS server is cold started, changes are made to the file system and to DCE. This section describes these changes in detail.

Effects on the file system

When an RQS server is cold started, the following directories are created:

- The **/opt/encinalocal/***cell_name***/server/***server_name* directory—the working directory for the RQS server. By default, it contains the following files: **keyfile**, **restart**, and **server.out**. In addition, it contains the **/logArchive** directory.
- The **/opt/encinamirror/***cell_name***/server/***server_name* directory—the mirror directory for the RQS server. By default, it contains the **restart.bak** file.

Note: On HP-UX machines, the working directory is **/var/opt/encina/local/***cell_name***/server/***server_name*, and the mirror directory is **/var/opt/encina/mirror/***cell_name***/server/***server_name*.

Effects on DCE

When an RQS server is cold started, several changes are made to the DCE Security Registry and to the CDS namespace. In addition, default ACLs and ACL entries are created.

Changes to the Security Registry: When an RQS server is cold started, the following changes are made to the Security Registry:

- The RQS server principal (*cell_name***/server/***server_name*) is created.
- An account is created for the RQS server principal.

Once these security objects are created, the keytab file is updated to include the RQS server principal and password.

Changes to CDS: When an RQS server is cold started, the CDS entry **/.:/***cell_name***/server/***server_name* is created.

Changes to ACLs: When an RQS server is cold started, two **/.:/***cell_name***/server/***server_name* ACLs are created. The first ACL controls access to the server itself. The following entries are added to this ACL:

- `{group encina_admin_group caxtq}`
- `{group encina_operator_group q}`

The second ACL controls access to the CDS directory entry for the RQS server. The following entries are added to this ACL:

- `{unauthenticated r--t---}`
- `{user` *cell_name*`/server/`*server_name* `rwdtc--}`
- `{group subsys/dce/cds-admin rwdtc--}`
- `{group subsys/dce/cds-server rwdtc--}`
- `{group encina_admin_group rwdtc--}`
- `{group encina_servers_group rwdtc--}`
- `{any_other r--t---}`

Modifying RQS ACLs

RQS server, queue, and queue set ACLs can be modified by using the **dcecp acl modify** command. The following sections provide detailed information about modifying RQS ACLs.

Modifying the RQS server ACL

Before modifying the ACL that controls access to an RQS server, ensure that you have the **administer** (**a**) permission on the server and that the cell manager is running with authorization enabled.

To modify an RQS server ACL, use the **dcecp acl modify** command. The following example adds a new entry to the ACL for the server called **rqs1**. The new entry grants members of the group **encina_admin_group** the **create** (**c**), **administer** (**a**), **destroy** (**x**), and **query** (**q**) permissions on the server.

```
% dcecp -c acl modify /.:/branch1/server/rqs1 \
-add '{group encina_admin_group caxq}'
```

Modifying a queue or queue set ACL

Before modifying the ACL of a queue or queue set, ensure that you have the **administer** (**a**) permission on the queue or queue set. (Note that you can also modify a queue or queue set ACL if you have the **administer** (**a**) permission on the RQS server itself.)

To modify an RQS queue or queue set ACL, use the **dcecp acl modify** command. Make sure that the name of the queue or queue set is prefixed with the **queue** or **qset** directory name, respectively. For example, the names of the queue **newAcctBilling** and the queue set **billing** in the server **/.:/branch1/server/rqs1** are as follows:

- **/.:/branch1/server/rqs1/queue/newAcctBilling**

- **/.:/branch1/server/rqs1/qset/billing**

The following example adds an entry to the ACL for the queue called **newAcctBilling**. The new entry grants the principal **client1** the **callback** (**l**), **dequeue** (**d**), **dequeue orphan** (**o**), **modify** (**m**), **read element** (**r**), and **query** (**q**) permissions on the queue.

```
% dcecp -c acl modify /.:/branch1/server/rqs1/queue/newAcctBilling \
-add '{user client1 ldomrq}'
```

You can specify default ACLs for all queues or queue sets by using the **-io** option with the **dcecp** command. The **-io** option allows you to manipulate the initial object ACL for the queue or queue set directory (the container of all queues or queue sets). For example, the following command adds an entry to the initial object ACL for the queue directory. The new entry grants the principal **client1** the **callback** (**l**), **dequeue** (**d**), **dequeue orphan** (**o**), **modify** (**m**), **read element** (**r**), and **query** (**q**) permissions for all new queues. Note that modifications to the initial object ACL affect only new objects; existing objects are not affected.

```
% dcecp -c acl modify /.:/branch1/server/rqs1/queue -io \
-add '{user client1 ldomrq}'
```

Controlling access to SFS objects

This section describes the ACLs used to control access to both SFS servers and SFS files. It also describes the changes made to the file system and to DCE when an SFS server is cold started.

The SFS server ACL

Each SFS server has an ACL governing access to it. The SFS server ACL is called **/.:/***cell_name***/server/***server_name*. Table 12 lists the possible permissions for an SFS server.

Table 12. ACL permissions for SFS files and servers

Permissions	For a Server	For a File
administer (**A**)	Modify the server's ACL. Use all **tkadmin** commands on SFS resources. Automatically grants **create** and **query** permissions on the server. Automatically grants **administer**, **exclusive open**, and **query** permissions on all files managed by the server.	Modify the file's ACL and its indices. Automatically grants **exclusive open** and **query** permissions for that file.

Table 12. ACL permissions for SFS files and servers (continued)

Permissions	For a Server	For a File
create (**C**)	Create and copy files at the server.	—
delete (**D**)	—	Delete records from the file.
exclusive open (**E**)	—	Open the file in exclusive mode.
insert (**I**)	—	Insert new records in the file.
query (**Q**)	Display information about the server.	List and query the file.
read (**R**)	—	Read records from the file.
update (**U**)	—	Modify records in the file.

When an SFS server is first started, the principal named as the exclusive authority is granted exclusive access to the server. If you are using Enconsole, the default exclusive authority is the principal **encina_admin**. Enconsole automatically grants full administrative permissions to the group **encina_admin_group** and then clears the exclusive authority.

If you are not using Enconsole, the exclusive authority must create entries on the server ACL. For example, the **administer** (**A**), **create** (**C**), and **query** (**Q**) permissions on the server can be granted by adding an appropriate entry to the server ACL. After permissions have been set, the exclusive authority must be cleared by using the **tkadmin clear exclusiveauthority** command.

Users with the **administer** (**A**) permission on an SFS server automatically acquire the **administer** (**A**), **exclusive open** (**E**), and **query** (**Q**) permissions on all files managed by the server and **create** (**C**) and **query** (**Q**) permissions on the server. Because the **administer** (**A**) permission grants access to all files managed by the server, it is recommended that you grant this permission cautiously.

The SFS file ACL

Each SFS file has an ACL that governs access to it. The SFS file ACL is called **/.:/***cell_name***/server/***server_name***/***file_name*. Table 12 on page 176 lists possible permissions for SFS files.

Users with the **administer** (**A**) permission on an SFS file automatically acquire the **exclusive open** (**E**) and **query** (**Q**) permissions on that file. Note that the creator of a file automatically gets full permissions (**A**, **D**, **E**, **I**, **Q**, **R**, and **U**) on the file.

For **sfsadmin** commands that modify a file or its indices, file permissions are checked first, followed by server permissions. Note that because the **administer** (**A**) permission on a server indirectly grants full file permissions, lack of file permissions does not necessarily prevent file-modification commands.

An example ACL for an SFS file follows. This ACL grants **query** (**Q**) and **read** (**R**) permissions to the principal **client1**, and it grants full file permissions to all members of the group **encina_admin_group**.

```
{user client1 ----QR-}
{group encina_admin_group ADEIQRU}
```

Effects of starting an SFS server

This section describes the changes that are made to the file system and to DCE when an SFS server is started for the first time.

Effects on the file system

When an SFS server is cold started, the following directories are created:

- The **/opt/encinalocal/***cell_name***/server/***server_name* directory—the working directory for the SFS server. By default, it contains the following files: **keyfile**, **restart**, and **server.out**. In addition, it contains the **/logArchive** directory.
- The **/opt/encinamirror/***cell_name***/server/***server_name* directory—the mirror directory for the SFS server. By default, it contains the **restart.bak** file.

Note: On HP-UX machines, the working directory is **/var/opt/encina/local/***cell_name***/server/***server_name*, and the mirror directory is **/var/opt/encina/mirror/***cell_name***/server/***server_name*.

Effects on DCE

When an SFS server is cold started, several changes are made to the DCE Security Registry and to the CDS namespace. In addition, default ACLs and ACL entries are created. The following sections describe the changes in detail.

Changes to the Security Registry: When an SFS server is cold started, the following changes are made to the Security Registry:

- The SFS server principal (*cell_name***/server/***server_name*) is created.
- An account is created for the SFS server principal.

Once these security objects are created, the keytab file is updated to include the SFS server principal and password.

Changes to CDS: When an SFS server is cold started, the CDS entry **/.:/***cell_name***/server/***server_name* is created.

Changes to ACLs: When an SFS server is cold started, two **/.:/***cell_name***/server/***server_name* ACLs are created. The first ACL controls access to the SFS server itself. The following entries are added to this ACL:

- `{group encina_admin_group ACQ}`
- `{group encina_operator_group Q}`

The second ACL controls access to the CDS entry for the SFS server. The following entries are added to this ACL:

- `{unauthenticated r--t---}`
- `{user` *cell_name*`/server/`*server_name* `rwdtc--}`
- `{group subsys/dce/cds-admin rwdtc--}`
- `{group subsys/dce/cds-server rwdtc--}`
- `{group encina_admin_group rwdtc--}`
- `{group encina_servers_group rwdtc--}`
- `{any_other r--t---}`

Modifying SFS ACLs

SFS server and file ACLs can be manipulated by using the **dcecp acl** command. The following sections provide detailed information about modifying SFS ACLs.

Modifying the SFS server ACL

To modify the ACL that controls access to an SFS server, you must have the **administer** (**A**) permission on the server. When modifying the SFS server ACL, remember that granting the **administer** (**A**) permission on the server indirectly grants access to all files managed by the server. Grant this permission cautiously.

The following example adds an entry to the ACL for the server **/.:/branch1/server/sfs1**, granting the user **user1** the **query** (**Q**) permission for the server:

```
% dcecp -c acl modify /.:/branch1/server/sfs1 \
-add '{user user1 Q}'
```

Modifying an SFS file ACL

To modify the ACL that controls access to an SFS file, you must have the **administer** (**A**) permission on the file. (Note that you can also modify SFS file ACLs if you have the **administer** (**A**) permission on the SFS server itself.)

The following example adds an entry to the ACL for the file **Inventory** (managed by the server **/.:/branch1/server/sfs1**), granting the principal **client1** the **query** (**Q**) permission on the file:

```
% dcecp -c acl modify /.:/branch1/server/sfs1/Inventory \
-add '{user client1 Q}'
```

This section describes the ACL used to control access to a PPC Gateway server. It also describes the changes made to the file system and to DCE when a PPC Gateway server is cold started.

The PPC Gateway server ACL

Each PPC Gateway server using DCE security has an ACL that controls access to the server. The PPC Gateway server ACL is named **/.:/***cell_name***/server/***server_name*. Table 13 lists possible permissions for PPC Gateway servers.

Table 13. ACL Permissions for PPC Gateway servers

Permissions	Usage
administer (**a**)	Allows principals (administrators) to execute **ppcadmin** commands and to view, modify, and create ACLs.
execute (**x**)	Allows PPC Executive applications to schedule conversations through the gateway server to logical unit (LU) 6.2 applications.

Note: For System Network Architecture (SNA)-to-Encina conversations, the gateway server requires Monitor **execute** (**x**) permission on PPC Executive applications in the Monitor to schedule conversations from LU 6.2 applications. The gateway server principal must be on the ACL for the LU name registered for the PPC Executive application or on the ACL for the individual transaction program name (TPN).

When a PPC Gateway server is first started, the principal named as the exclusive authority is granted exclusive access to the server. If you are using Enconsole, the default exclusive authority is the principal **encina_admin**. Enconsole automatically grants full administrative permissions to the group **encina_admin_group** and then clears the exclusive authority.

If you are not using Enconsole, the exclusive authority must create entries on the server ACL. For example, the **administer** (**a**) and **execute** (**x**) permissions on the server can be granted by adding an appropriate entry to the server ACL. After permissions have been set, the exclusive authority must be cleared by using the **tkadmin clear exclusiveauthority** command.

The following is an example of an ACL for a gateway server. The first ACL entry grants the principal **encina_ppcexec_app1** the PPC **execute** (**x**)

permission to the gateway server; the second ACL entry grants members of the group **encina_admin_group** the PPC **administer** (**a**) and **execute** (**x**) permissions to the gateway server.

```
{user encina_ppcexec_app1 -x}
{group encina_admin_group ax}
```

Effects of starting a PPC Gateway server

This section describes the changes that are made to the file system and to DCE when a PPC Gateway server is started for the first time.

Effects on the file system

When a PPC Gateway server is cold started, the following directories are created:

- The **/opt/encinalocal/***cell_name***/server/***server_name* directory—the working directory for the PPC Gateway server. By default, it contains the following files: **keyfile**, **restart**, and **server.out**. In addition, it contains the **/logArchive** directory.
- The **/opt/encinamirror***cell_name***/server/***server_name* directory—the mirror directory for the PPC Gateway server. By default, it contains the **restart.bak** file.

Note: On HP-UX machines, the working directory is **/var/opt/encina/local/***cell_name***/server/***server_name*, and the mirror directory is **/var/opt/encina/mirror/***cell_name***/server/***server_name*.

Effects on DCE

When a PPC Gateway server is cold started, several changes are made to the DCE Security Registry and to the CDS namespace. In addition, default ACLs and ACL entries are created. The following sections describe the changes in detail.

Changes to the Security Registry: When a PPC Gateway server is cold started, the following changes are made to the Security Registry:

- The PPC Gateway server principal (*cell_name***/server/***server_name*) is created.
- An account is created for the PPC Gateway server principal.

Once these security objects are created, the keytab file is updated to include the PPC Gateway server principal and password.

Changes to CDS: When a PPC Gateway server is cold started, the CDS entry **/.:/***cell_name***/server/***server_name* is created.

Changes to ACLs: When a PPC Gateway server is cold started, two **/.:/***cell_name***/server/***server_name* ACLs are created. The first ACL controls access to the PPC Gateway server itself. The following entries are added to this ACL:

- `{group encina_admin_group ax}`

- {group encina_operator_group a}

The second ACL controls access to the CDS entry for the PPC Gateway server. The following entries are added to this ACL:

- {unauthenticated r--t---}
- {user *cell_name*/server/*server_name* rwdtc--}
- {group subsys/dce/cds-admin rwdtc--}
- {group subsys/dce/cds-server rwdtc--}
- {group encina_admin_group rwdtc--}
- {group encina_servers_group rwdtc--}
- {any_other r--t---}

Modifying the PPC Gateway server ACL

If security is enabled for a gateway server, you must add entries to the gateway server ACL to give principals access to it. You must have the PPC **administer** (**a**) permission on the gateway server ACL to add an entry to the ACL.

Note: Because users with **administer** (**a**) permission can grant anyone access to the server, grant the **administer** (**a**) permission cautiously.

To modify the ACL that controls access to the gateway server, use the **dcecp acl modify** command. The following example adds two entries to the ACL for the **gwy1** gateway server. The first entry grants members of the group **encina_admin_group** the PPC **administer** (**a**) and **execute** (**x**) permissions to the gateway server; the second entry grants the principal **encina_ppcexec_app1** only the PPC **execute** (**x**) permission to the gateway server.

```
% dcecp -c acl modify /.:/branch1/server/gwy1 \
-add '{{group encina_admin_group ax} {user encina_ppcexec_app1 x}}'
```

Chapter 9. Modifying server behavior

This chapter covers performance-related and other topics for Encina servers. It describes how to optimize space usage in Recoverable Queueing Service (RQS) servers, how to adjust performance parameters in SFS servers and DE-Light gateways, and how to modify the number of processing agents (PAs) for Monitor application servers. It also describes how to use server sets, schedules, Fast Local Transport (FLT), and dynamic resource manager management.

Improving space usage in an RQS server

You can optimize space usage in an RQS server by doing any of the following:

- Adjusting the number of buckets in a server key table in response to usage patterns (using the **rqsadmin reorganize keytable** command).
- Specifying when to make adjustments to the keytable (using the **rqsadmin set keytablepolicy** command).
- Eliminating wasted space in server queues (using the **rqsadmin reorganize queues** and **rqsadmin set pagecleanup** commands).
- Displaying information about space usage in the Recoverable Storage Allocator (RSA) (using the **rqsadmin query rsa** command).
- Reclaiming space in the RSA (using the **rqsadmin shrink rsa** command).

These commands are described in the following sections.

Optimizing space usage in the server key table

When an RQS server is first started, Encina provides a default value for the number of buckets in the server key table. A *bucket* is a repository for one or more keys in the key table. You can change the default value to reflect expected capacity and usage by modifying it on the RQS Advanced Options form in Enconsole or by specifying a value for the **-buckets** option of the **rqsadmin init rqsservice** command (when starting a server from the command line). The number of buckets can be adjusted over time so that it represents the best use of space based on actual usage patterns. Also, you can set policies for when to increase or decrease the number of buckets. The **rqsadmin reorganize keytable** command adjusts the number of buckets in the server key table. The **rqsadmin set keytablepolicy** command assigns a policy that governs when and how the key table is organized.

Adjusting the number of key table buckets

As necessary, you can increase or decrease the number of buckets in a server key table to keep bucket usage at an optimal level. The **rqsadmin reorganize keytable** command adjusts the key table by using either of two reorganization methods—**expand** and **shrink**. The **expand** method doubles the number of buckets in a key table. The **shrink** method halves the number of buckets in a key table.

The complete syntax of the **rqsadmin reorganize keytable** command is

```
rqsadmin reorganize keytable -server server_name {expand | shrink}
```

The following command doubles the number of buckets in a server key table:

```
% rqsadmin reorganize keytable expand
```

Specifying when to reorganize a server key table

Each reorganization method has an associated policy for when that method is to be used for the key table. Valid policies are **manual** and **auto** (for automatic). The default policy for both the expand and shrink methods is **manual**—that is, the key table is reorganized using the specified method only when you issue the **rqsadmin reorganize keytable** command. If the policy is automatic, RQS automatically reorganizes the key table according to the percent of total buckets in use. If more than 70 percent of the total number of buckets are in use, the **expand** method is used. If less than 20 percent of the buckets are in use, the **shrink** method is used. The **rqsadmin set keytablepolicy** command specifies, for each reorganization method, the policy to be used for that method.

Note that the **rqsadmin reorganize keytable** command is independent of the reorganization policy (automatic or manual) in effect for the key table. In other words, if the policy for a given reorganization method is automatic, the **rqsadmin reorganize keytable** command overrides the policy and reorganizes the key table manually. The automatic policy resumes after the command is issued.

The complete syntax of the **rqsadmin set keytablepolicy** command is

```
rqsadmin set keytablepolicy -server server_name {expand | shrink} {manual | auto}
```

The following command sets the key table policy for the shrink method to automatic:

```
% rqsadmin set keytablepolicy shrink auto
```

Improving space usage for queues

A queue consists of a header and a body. A *queue body* is a linked list of pages that contains pointers to the physical location of queue elements. By default, queue elements are stored in the body pages of queues if possible. Some queue operations, such as requeueing and dequeueing by key or by element

ID, can leave gaps in the body pages of queues. You can eliminate wasted space in queues by using the **rqsadmin reorganize queues** and **rqsadmin set pagecleanup** commands. The **rqsadmin reorganize queues** command reclaims unused space in the body pages of queues. The **rqsadmin set pagecleanup** command specifies threshold values for triggering and stopping automatic page cleanup for queues. These commands are described in the following sections.

Reclaiming page space for queues

The **rqsadmin reorganize queues** command reclaims gaps in page space caused by dequeueing and requeueing. By default, the command reclaims space in all queues in the server. You can also reclaim space in a single queue or in all queues of a queue set.

The **rqsadmin reorganize queues** command provides three options for reclaiming space in queue body pages: **clean**, **compress**, and **fullcompress**. The three options use increasingly aggressive methods of reclaiming space. Each option represents an increase in computational requirements and in the amount of space that is reclaimed.

- The **clean** option is the simplest and least expensive option. It reclaims unused space in queues by freeing up empty body pages and space occupied by orphaned elements that were not requeued.
- The **compress** option reclaims space in the same way, but also compresses body pages (compacts pages that contain gaps left by dequeued elements).
- The **fullcompress** option does the same thing as the **compress** option, but also attempts to move elements for more efficient space utilization. This option requires the most computational resources and should be used only when necessary.

The complete syntax of the **rqsadmin reorganize queues** command is

```
rqsadmin reorganize queues -server server_name [-qsetname queue_set_name]
[-queuename queue_name] {clean | compress | fullcompress}
```

The following command reorganizes all server queues using the compress option:

```
% rqsadmin reorganize queues compress
```

Setting page cleanup behavior for queues

The **rqsadmin set pagecleanup** command specifies threshold values for triggering and stopping automatic page cleanup for queues. Page cleanup improves space utilization and can result in faster element access. The cleanup operation is the same as the **clean** option available through the **rqsadmin reorganize queues** command. Cleanup consists of deleting orphaned elements that have not been requeued and reclaiming all resulting unused space.

A page cleanup daemon automatically attempts to clean up pages when the number of empty pages in a queue falls within a range of predefined high and low thresholds. Note that this command controls when *automatic* cleanup is done. Page cleanup can also be done manually by specifying the **clean** argument to the **rqsadmin reorganize queues** command.

The complete syntax of the **rqsadmin set pagecleanup** command is

```
rqsadmin set pagecleanup -server server_name [-pagehigh page_high]
[-pageratiohigh page_ratio_high] [-pagelow page_low]
[-pageratiolow page_ratio_low] [-sleep sleep] [-maxsleep maximum_sleep_units]
```

A cleanup request is triggered when the number of queue pages exceeds the page high *and* the page ratio high values. The *page_high* argument specifies the highest number of empty queue pages that trigger a cleanup request. The default value is 10. The *page_ratio_high* argument specifies the highest percentage of queue pages that can be empty before a cleanup request is triggered. The default value is 20.

If the number of empty pages falls below the page low value *or* the page ratio low value, cleanup requests cease until the level of empty pages again rises above the page low threshold. The *page_low* argument specifies the lowest number of empty pages that trigger a cleanup request for a queue. The default value is 5. The *page_ratio_low* argument specifies the lowest percentage of queue pages that can be empty before triggering a cleanup request for a queue. The default value is 10.

If the page cleanup operation fails to reclaim enough pages for the queue (that is, if it fails to reduce the number of empty pages to a level below the low thresholds, signalling the page cleanup to stop), page cleanup is retried in 15 seconds (the default value of the **-sleep** option). The sleep timer doubles the number of seconds to wait before the next page cleanup, up to a maximum of 600 seconds (the default value of the **-maxsleep** option).

There are performance costs in running the daemon cleanup process. A higher setting for the *page_high* and *page_ratio_high* values (and a lower setting for *page_low* and *page_ratio_low* values) causes the daemon to run more frequently. A lower setting for the *page_high* and *page_ratio_high* values causes the daemon to run less frequently. When the cleanup process runs less frequently, however, the dequeueing operation can suffer if too many empty pages remain. (The dequeueing operation scans a queue to locate an element and unnecessarily scans several empty pages before locating the element.) In general, the default settings for the **rqsadmin set pagecleanup** command options provide a reasonable trade-off between the cost of running the cleanup daemon, efficient dequeueing, and space conservation.

In general, if an application's performance does not depend on the sequential dequeueing operation and there is sufficient disk space to tolerate some empty pages, the thresholds can be set to reduce the frequency of cleanups. You can then issue a **rqsadmin reorganize queues** command at nonpeak hours to force page cleanup. If the application's performance depends on the sequential dequeueing operation or cannot afford disk space with empty pages, the thresholds can be set so that the daemon process runs more frequently.

Note that both the *page_high* and *page_ratio_high* values must be met to trigger the page cleanup daemon. It is possible that queues with a small number of pages never meet both criteria. To manually reclaim space in small queues, issue a **rqsadmin reorganize queues** command with the **clean** argument.

The following command specifies 15 as the highest number of pages that trigger a page cleanup request. It also increases the initial sleep time before the next attempt at page cleanup:

```
% rqsadmin set pagecleanup -pagehigh 15 -sleep 20
```

Reclaiming space in the RSA of an RQS server

You can reclaim space and display information about space usage in the RSA of an RQS server. The RSA stores element types, queue elements that are more than a page in size, and access control lists (ACLs). The initial percentage of volume allocation for the RSA is specified when you define the server. The space allocated by the RSA can grow from the initial allocation to the limit of the data volume. This can prevent other storage allocators (such as the queue page allocator) from growing. When the volume limit is reached or is nearly full, you must reclaim freed space in the RSA. The **rqsadmin query rsa** command displays the amount of space used by the RSA and provides an estimate of the amount of space that can be reclaimed by the **rqsadmin shrink rsa** command.

Displaying information about space usage in the RSA

If an RQS volume is performing poorly, it can possibly be running out of space. Enqueue failure messages can also indicate a lack of space. The **rqsadmin query rsa** command checks the total pages and the total pages used in the RSA so that you can determine whether a full RQS volume is the cause of the problem. It also indicates the number of pages eligible for shrinking (reclaiming).

The complete syntax of the **rqsadmin query rsa** command follows:

rqsadmin query rsa -server *server_name* [**-verbose**]

When this command is used without the **-verbose** option, the command output includes only the number total pages and pages used. When this command is used with the **-verbose** option, more detailed command output is displayed.

The following command requests detailed information about the RSA of an RQS server. If the command output indicates that there are pages eligible for shrinking (reclaiming), use the **rqsadmin shrink rsa** command. The output of the following **rqsadmin query rsa** command shows that 1677 pages out of 10,225 allocated pages of the RSA are used and that each RSA pool has 1536 shrinkable pages:

```
% rqsadmin query rsa -verbose
used pages:              1677    (16%)
merge pages:              26      (0%)
total pages:             10255
     Merge Daemon Parameters:
          retry initial sleep time(seconds):                3
          retry maximum sleep time(seconds):                60
          minimum merge list entries to trigger daemon:     30
          minimum merge list entries to stop daemon:        10
     Overhead Pages(Area Id: 4):     15
     Total Pools:            5
     Pool :1(AreaId: 5)
          merge triggered:           FALSE
          merge list pages:          2       (0%)
          merge list entries:        1
          free list pages:           1679    (81%)
          free list entries:         18
          used pages:                367     (17%)
          total pages:               2048
          initial pages:             512
          minimum shrink size (in pages): 256
          shrinkable pages: 1536
     Pool :2(AreaId: 6)
          merge triggered:           FALSE
          merge list pages:          2       (0%)
          merge list entries:        1
          free list pages:           1753    (85%)
          free list entries:         23
          used pages:                293     (14%)
          total pages:               2048
          initial pages:             512
          minimum shrink size (in pages): 256
          shrinkable pages: 1536
     Pool :3(AreaId: 7)
          merge triggered:           FALSE
          merge list pages:          0       (0%)
          merge list entries:        0
          free list pages:           1727    (84%)
          free list entries:         21
          used pages:                321     (15%)
          total pages:               2048
```

```
     initial pages:             512
     minimum shrink size (in pages): 256
     shrinkable pages: 1536
Pool :4(AreaId: 8)
     merge triggered:           FALSE
     merge list pages:          0       (0%)
     merge list entries:        0
     free list pages:           1713    (83%)
     free list entries:         21
     used pages:                335     (16%)
     total pages:               2048
     initial pages:             512
     minimum shrink size (in pages): 256
     shrinkable pages: 1536
Pool :5(AreaId: 9)
     merge triggered:           FALSE
     merge list pages:          22      (1%)
     merge list entries:        4
     free list pages:           1680    (82%)
     free list entries:         9
     used pages:                346     (16%)
     total pages:               2048
     initial pages:             512
     minimum shrink size (in pages): 256
     shrinkable pages: 1536
```

See the **rqsadmin query rsa** reference page for a more detailed explanation of the command output.

Reclaiming space in the RSA

The **rqsadmin shrink rsa** command reclaims allocated but unused space so that other allocators (such as the queue page allocator) can expand to fill available volume space. In contrast, the **rqsadmin reorganize queues** command reclaims space in the queue page allocator; it compresses gaps left on queue pages due to requeueing and dequeueing by key or by element ID.

In most cases, it is not necessary to use the **rqsadmin shrink rsa** command. Versions of Encina prior to version 2.0 stored all RQS elements in the RSA. In servers defined according to the earlier storage model, the **rqsadmin shrink rsa** command can be necessary to manage space usage by removing elements smaller than one page. However, as of Encina version 2.0, the RSA stores only elements larger than one page.

The complete syntax of the **rqsadmin shrink rsa** command follows:

```
rqsadmin shrink rsa -server server_name
```

The following command shrinks the RSA of an RQS server:

```
% rqsadmin shrink rsa
```

Tuning an SFS server

Many factors can affect the performance of an SFS server. In addition to platform and environment considerations, the following can impact server performance:

- Chunk size
- Buffer pool size
- Thread pool sizes
- Checkpoint interval

Note that many of the above server attributes have default values. The default values can be changed when a server is initially started by editing the attribute's value on the appropriate Enconsole form (or by using command-line options of the **sfs** command).

This section describes how chunk size impacts disk space usage and performance and how to choose chunk size. It also describes the default settings for the other server attributes listed above.

Specifying chunk size for physical volumes

When you create a physical volume to back an SFS logical volume, you must specify the chunk size for the physical volume. *Chunk size* is an allocation unit used by the SFS. Chunk size must be specified in units of pages, and it must be a power of 2. Note that ″page″ refers to the Encina page size—4096 bytes. Chunk size is important to the SFS because the chunk size specified for a volume is the increment by which the SFS allocates disk space when a file or index is expanded.

Chunk size can impact the following:

- How efficiently disk space is used
- The amount of system overhead incurred when allocating space

If a chunk size is large compared to the size of a disk, the SFS may not make maximum use of available disk space when storing files or indices on the volume. Small chunk sizes typically allow better use of disk space. However, with small chunk sizes, the allocation of disk space can occur more often than necessary (for example, when a file requiring a large amount of disk space grows rapidly). Every incremental allocation of disk space represents system overhead. If file space requirements are known in advance, you can avoid frequent space allocation by preallocating disk space when the file or index is created (with the **-preallocate** option of the file or index creation command).

Typically, neither of these costs—unused disk space or system overhead—is substantial. It is recommended that you set the initial chunk size to 64 pages; you can alter chunk size on subsequent volumes if needed. If you are using

Enconsole to create physical volumes, Enconsole uses the default chunk size of 64. If you are using **tkadmin** commands to create physical volumes, you specify chunk size as an argument to the **tkadmin create pvol** command.

Specifying buffer pool size

The *buffer pool size* is the amount of main memory used as a cache for accessing recently used portions of Encina volumes. A larger buffer pool generally reduces file system disk I/O but requires more virtual memory. Using too small a buffer pool may result in excessive SFS disk I/O. Using too large a buffer pool may result in paging or swapping, which usually has a more detrimental effect on performance.

The buffer pool size is expressed in blocks of 1024 bytes. The default buffer pool size is 1000 kilobytes. You can modify the default buffer pool size when an SFS server is started by changing its value on the Recovery Options form (if using Enconsole) or by using the **-b** option of the **sfs** command (if starting a server from the command line).

Specifying thread pool sizes

A thread pool size determines the number of concurrent operations that can run. You can specify the sizes of the thread pools to be used by an SFS server. There are two thread pools: *processing* and *emergency*. The *processing* pool is used for normal processing and administrative operations. The *emergency* pool is used for operations that query and free resources. Examples of operations that query and free resources are the **sfs_AbortOperation**, **sfs_CloseServerOfd**, **sfs_ListServerOfds**, **sfs_ListTransactionLocks**, and **sfs_GetServerOfdInfo** functions.

You can modify the default thread pool sizes by changing the default values on the SFS Advanced Options form (if using Enconsole), or by using the **-P** option of the **sfs** command (if starting a server from the command line). When using the **-P** option, the *thread_pool_sizes* argument is specified as a single value or two colon-separated values (*processing***:***emergency*). If a single value is specified, that value indicates the size of the *processing* pool. If no *processing* size is specified, the processing calls will be handled by the default thread pool. The size of the default thread pool is determined by the value of ENCINA_TPOOL_SIZE. If that variable is not defined, the default is 5. If no *emergency* size is specified, the default value is 0.

Specifying checkpoint interval

The *checkpoint interval* is the interval of time between checkpoints made by an SFS server. Frequent checkpointing can reduce restart time but decreases overall performance.

You can modify the default checkpoint interval when an SFS server is started by changing its value on the Recovery Options form (if using Enconsole), or by using the **-i** option of the **sfs** command (if starting a server from the

command line). When using the **-i** option, the checkpoint interval is specified as 3 numbers separated by colons—*seconds*:*minimum records*:*maximum records*. At the end of every interval of *seconds*, a checkpoint is taken if the number of log records written since the previous checkpoint exceeds the value of *minimum records*. Further, a checkpoint is taken as soon as the number of log records written since the previous checkpoint exceeds *maximum records*, regardless of the value of *seconds*. The default values are 60:5000:100000. These values can be changed. You must specify all three values whether you are changing one or all of the values. For example 60:10000:100000 changes the value of *minimum records* to 10000.

Tuning a DE-Light Gateway

Factors such as timeout values, thread pool specifications, and the number of client connections can affect the performance of a DE-Light Gateway. This section describes how to tune DE-Light Gateway performance by using environment variables and options to the **drpcadmin** and **drpcgwy** commands.

Setting the maximum transaction timeout value

You can use the **drpcadmin set trantimeout** command to specify a maximum timeout value for all transactions processed through the gateway. The timeout value is a number of seconds specified as an integer. The maximum gateway timeout takes precedence over any timeouts specified by the client application; however, other Encina timeouts can influence when a transaction actually times out.

A transaction's running time is measured from the time the transaction begins in the gateway. Generally, a transaction begins at the gateway with the first TRPC the client makes. If the maximum gateway timeout value is reached, regardless of the client timeout values, the gateway aborts the transaction.

The complete syntax for the command follows:

```
drpcadmin set trantimeout max_timeout [-gateway gateway_name]
```

where

- *max_timeout* specifies the maximum timeout value for a TRPC. The *max_timeout* argument is an integer value set in seconds. A value of **0** (zero) disables the timeout (allows a transaction to wait for an unlimited period of time). The default value is **180**.
- **-gateway** *gateway_name* specifies the DCE CDS entry or binding string used to identify a gateway.

For example, the following command sets the maximum timeout value for transactions processed through the **/.:/encina/server/gate1** gateway to **10** seconds:

```
% drpcadmin set trantimeout 10 -gateway /.:/encina/server/gate1
```

You can check the current maximum timeout value by using the **drpcadmin query gateway** command.

Using environment variables

You can use the following environment variables when administering a DE-Light gateway. These variables provide a convenience for often-used arguments for the **drpcadmin** and **drpcgwy** commands.

The DRPC_GATEWAY environment variable is used with the **drpcadmin** command. It specifies the DCE CDS entry or binding string used to identify the gateway. If you specify a gateway by using this variable, you can omit the **-gateway** option when administering that gateway by using the **drpcadmin** command. If you specify a gateway with the **-gateway** option, that value overrides the value set by this variable. The **drpcgwy** command does not recognize this variable.

The DRPC_MAX_CONNECTIONS environment variable specifies the number of DE-Light client connections that the gateway can handle.

The DRPC_GATEWAY_TUNING environment variable can be used to set the following parameters for a DE-Light Gateway:

- `bindingCacheSize`—Specifies the number of server bindings to maintain in the gateway's per-session binding handle cache. The oldest cached bindings are freed as soon as this number of entries has been exceeded. The default value is **10** entries.
- `rpcComTimeout`—Specifies how long to wait when a server is first contacted. It takes an integer argument with a value between **0** (zero) and **10**, where **0** means wait the minimum amount of time specified by the network protocol, and **10** means wait forever. Note that the value does *not* represent seconds; it represents a relative amount of time to spend to establish a client/server relationship (binding). The default value is **5**.
- `rpcServerComTimeout`—Specifies how long to wait on each communications attempt. It takes an integer argument with a value between **0** (zero) and **10**, where **0** means wait the minimum amount of time specified by the network protocol, and **10** means wait forever. Note that the value does *not* represent seconds; it represents a relative amount of time to spend for each communications attempt. The default value is **5**.
- `minTpoolSize` and `maxTpoolSize`—The DE-Light gateway grows and shrinks the number of threads prepared to service dynamic remote procedure calls (DRPCs) on behalf of client sessions. These parameters specify the lower and upper boundaries of the thread pool size. The default values are **4** threads and **64** threads, respectively.

- tpool_idle_time_threshold—Controls the amount of time that a DRPC service thread remains idle before being terminated (down to a minimum number of minTpoolSize). The default value is **120** seconds.
- tpool_busy_threshold—Specifies the number of pending requests required before spawning an additional thread to service DRPCs. The default value is **1** (request).
- tpool_busy_threshold_interval—Specifies the period of time, in seconds, for which all worker threads can be busy before a new thread is created. A large number causes the gateway to create worker threads essentially on demand. The default value is **2147483647** (seconds).
- session_LoginCacheLifetime—Specifies the number of seconds that login information associated with an involuntarily closed TCP or idle HTTP session is maintained in the gateway resident login cache table after its last use. The default value is **1800** (seconds).
- com_MaxHttpCnctns—Limits the total number of active HTTP-based sessions. This parameter differs from the comHttpMaxConnect parameter in that it affects both current connections with open sockets and those that do not have open sockets. The default maximum is **512** HTTP connections.
- sslV2SessionTimeout and sslV3SessionTimeout—Specifies the maximum amount of time, in seconds, that an SSL session can be reestablished. The SSL protocol has a provision that allows clients that are reconnecting to a server to skip the public key handshake, thus improving performance. Restricting the time during which a session can be reused limits the vulnerability to attack and limits the amount of information that must be maintained for old sessions. The default values are **100** (seconds) and **43200** (seconds), respectively.
- com_TcpBacklogSize and com_HttpBacklogSize—Specifies the size of the socket **listen()** call backlog queue for the respective protocols. Connections are queued up to this limit; when the limit is reached, connections fail, and an ECONNREFUSED socket error results. The default value for each parameter is **32** connections.
- com_HttpSocketTimeout and com_TcpSocketTimeout—Specifies the amount of time that the respective socket I/O operations block. This timeout prevents clients that perform poorly from preventing service to other clients by interfering with network I/O completion. The default value for each parameter is **300** seconds.
- com_TcpMaxConnect—Specifies the maximum number of open TCP/IP connections. The default is **256** active TCP/IP connections. You can override the value of this tuning parameter by using the DRPC_MAX_CONNECTIONS environment variable.
- com_HttpMaxConnect—Specifies the maximum number of open HTTP connections. This parameter differs from com_MaxHttpCnctns in that it constrains active HTTP connections with open sockets. The default value is **256** connections.

- com_SocketBufferSize—Specifies the value for TCP SO_SNDBUF and SO_RCVBUF socket option parameters, which specify the size of network send and receive buffers. The size can be increased for applications that perform RPCs with large data sizes (0 = system defaults). The default value is **0** (bytes).
- com_TcpNoDelay—Controls the socket option setting to delay sending of data to combine multiple packets. Turning on TCP_NODELAY prevents a delay before sending a packet, but does not make efficient use of network resources, and has its own problems handling large messages over networks with high latency. The default value is **0** (TCP_NODELAY disabled).
- com_HttpKeepAlive—Specifies the number of seconds to keep a socket open for additional HTTP requests from the same client. A setting of **0** disables keepAlive. The default value is **60** seconds.

Set the DRPC_GATEWAY_TUNING environment variable by using the following format:

"parameter1=value1[*,parameter2=value2...*]"

For example, to set the values of the rpcComTimeout and rpcServerComTimeout parameters to **7** and **2**, respectively, issue the following command in a C shell:

```
setenv DRPC_GATEWAY_TUNING "rpcComTimeout=7,rpcServerComTimeout=2"
```

Modifying the number of processing agents

Processing agents (PAs) are instances of server code that handle multiple requests for services. As the number of PAs associated with a server increase, more clients can access the server but more system resources are consumed. This section shows you how to change the number of PAs for a Monitor application server. The changes take effect immediately.

Perform the following steps to modify the number of PAs for a Monitor application server:

1. Choose the **Query/Modify Server** command from the **Actions** menu, and then choose your Monitor application server from the Selection box. Enconsole displays the Monitor Application Server form.
2. Choose the **MAS Advanced Options** button, and in the **Number of PAs** field, type the new number of processing agents for your server. You can also modify other attributes of the server from the MAS Advanced Options form. Figure 55 on page 196 shows an example.

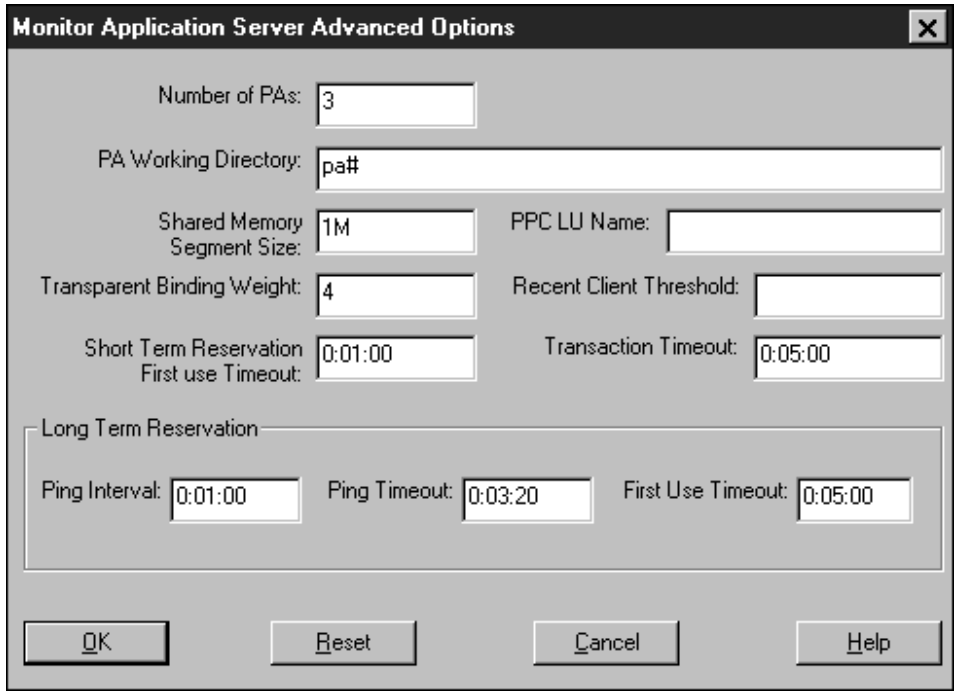

Figure 55. MAS Advanced Options form

3. When you have finished modifying the server's attributes, choose the **OK** button on the Monitor Application Server Advanced Options form, and then choose the **OK** button on the Monitor Application Server form. Enconsole stores the changes and displays a message in the Command Status display screen.
4. Choose the **OK** button to close the Command Status display screen.

   Note that when you *increase* the number of PAs, Enconsole displays the state of the PAs that are starting on the Encina Console display screen. Figure 56 on page 197 shows the messages that are displayed when the number of PAs for the server **Jmas** is increased. When all PAs for a Monitor application server are running, Enconsole displays the state of the Monitor application server only—not the state of the individual PAs.

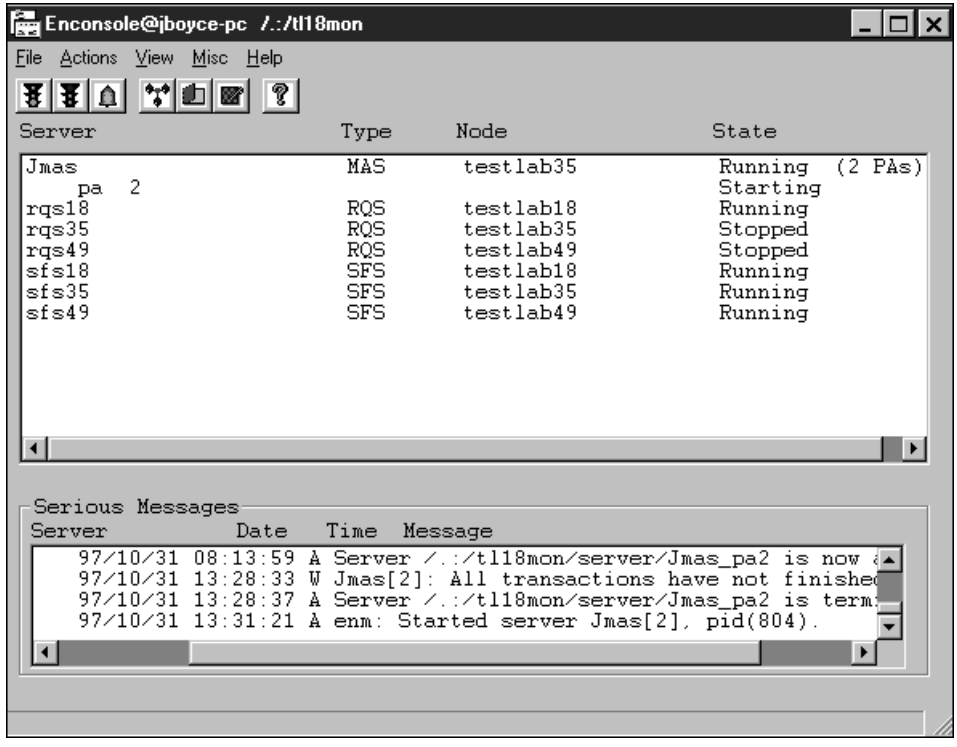

Figure 56. Encina Console PA startup messages

Similarly, when you *decrease* the number of PAs, the Serious Messages display screen shows which PAs are being terminated. Figure 57 shows the messages that are displayed when the number of PAs for the **Jmas** server is reduced from 6 to 2.

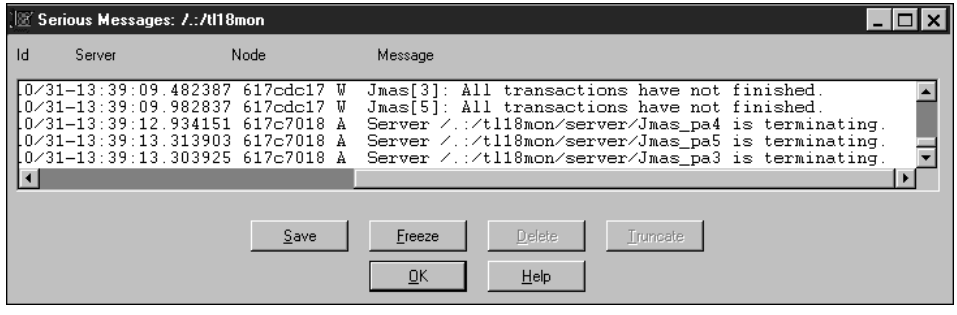

Figure 57. Serious Messages PA termination messages

Using server sets

A *server set* is a collection of servers that can be started and stopped as a unit. Server sets simplify system administration by enabling you to start multiple servers simultaneously. When a server is a member of a server set, it can be started or stopped independently of other servers in the set, and can be a member of multiple sets.

Defining a server set

Enconsole provides a Server Set form to define server sets (Figure 58).

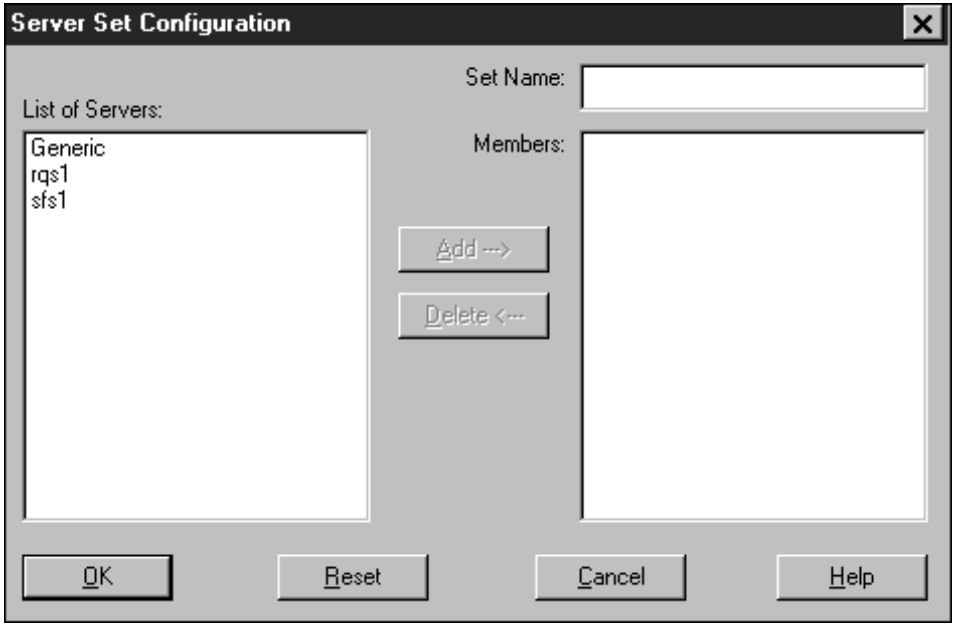

Figure 58. Server Set form

Perform the following steps to define a server set:

1. Choose the **Define Server Set** command from the **Actions** menu. Enconsole displays the Server Set form.
2. Type the name of the server set you want to define in the **Name** field.
3. Select a server to add to the server set from the **List of Servers**, and then choose the **--->** button. Enconsole adds the selected server to the server set and removes the selected server from the **List of Servers**. You can continue to select servers and use the **--->** button to add servers to the server set. To remove a selected server from a server set, select the server from the **Members** field and choose the **<---** button.

4. When you have finished adding servers to the server set, choose the **OK** button on the Server Set form. Enconsole stores the server set and displays the Command Status display screen. Choose the **OK** button to close the Command Status display screen.

Starting a server set

Use the **Start Server Set** command to start the servers in a server set that are not currently running. Before you can start servers in a server set, the following conditions are required:

- The server must have already been cold-started. Use the **Start Server** command to start a server for the first time before attempting to start the server as a member of a server set. (The **Start Server Set** command does not create resources such as the server principal, keytab file, and working directory for a server in a set. These resources are essential to the operation of the server and are created when the server is started for the first time using the **Start Server** command.)
- The node manager must be running.

You can also use a schedule to start a server set. (See "Using schedules to start and stop servers" for information about schedules.)

Perform the following steps to start a server set:

1. Choose the **Start Server Set** command from the **Actions** menu, and then choose your server set from the Selection box. Enconsole retrieves the server set and displays the Command Status display screen.
2. When Enconsole displays a message to show that the command was successful, choose the **OK** button to close the Command Status display screen.

Using schedules to start and stop servers

A schedule is a procedural plan that specifies times at which to start and stop a server or server set. You can use schedules to establish one-time or periodic events that happen automatically, without further intervention. For example, you can start a group of servers every morning and stop them every evening, or you can run a group of maintenance servers one day a week.

Enconsole provides a Schedule form (Figure 59 on page 200) to define schedules.

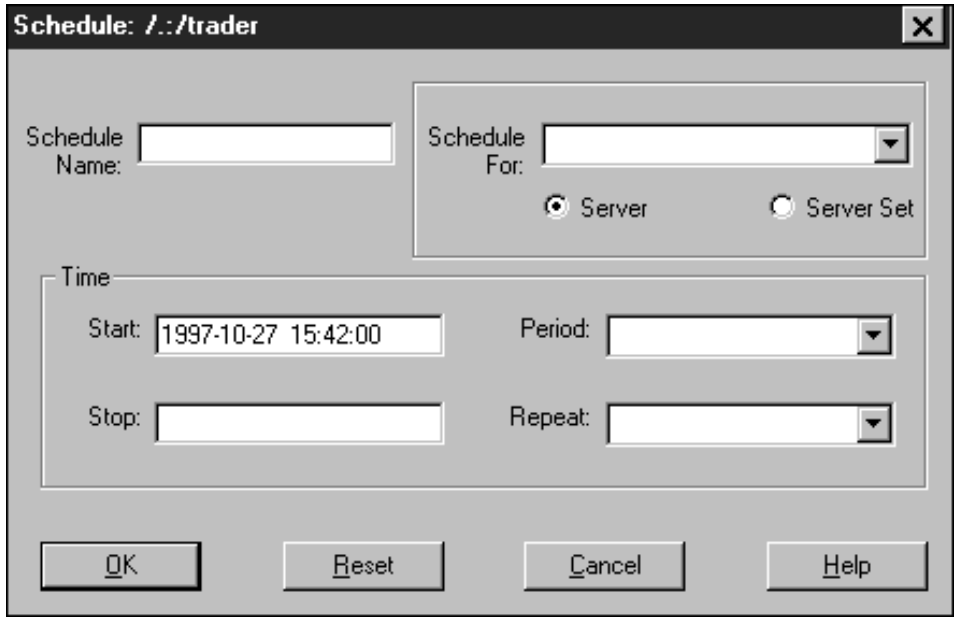

Figure 59. Schedule form

Start and stop times

You can specify start times and stop times for both one-time events and periodic events. The **Start** field specifies the initial time at which a server or server set is to be started—if a start value is not specified, the schedule does not start the server. The **Stop** field form specifies the initial time at which the server is to be stopped—if a stop value is not specified, the schedule does not stop the server. You must specify a start time *or* a stop time; you do not have to specify both. The start time should exceed the current date and time. If you specify both a start time and a stop time, the stop time must exceed the start time.

Enconsole accepts the following formats for initial start and stop times:

- **Dates** — Enconsole requires only the month and the day of the month. Table 14 shows some of the date formats that Enconsole accepts. If you do not specify a day, Enconsole uses the current day. If you do not specify a year, Enconsole uses the current year.

Table 14. Date format

7/14
July 14
Jul 14

- **Times** — Enconsole accepts a time (in *HH*:*MM*:*SS* format, where *HH* is the number of hours, *MM* is the number of minutes, and *SS* is the number of seconds) combined with a date. As mentioned earlier, if you do not specify a date, Enconsole uses the current date. Table 15 shows some of the time formats that Enconsole accepts. The hours value must be between 0 and 23. The minutes and seconds values must be between 0 and 59. If you do not specify a time value, Enconsole uses 00:00:00.

Table 15. Initial start and stop time format

13:30:00
1:30 PM

When you exit a **Start** or **Stop** field, Enconsole converts your input to the following format:

```
Day_of_Week Month Day_of_Month, HH:MM:SS Year
```

For example, Enconsole converts the value **7/14/95 1:00 PM** to the following format:

```
Fri Jul 14, 13:00:00 1995
```

Periodic events

If you specify a value for the **Repeat** field or the **Period** field, you define a periodic event. If you do not specify a repeat or period value, you define a one-time event. To indicate that a schedule should be performed a specified number of times, use the **Repeat** field. You can specify an integer value, or you can choose the value **Forever** to indicate that the schedule should be repeated indefinitely.

To specify an interval at which a schedule should be performed, use the **Period** field. If a repeat value is specified, a period is required. You can choose an **Hourly**, **Daily**, or **Weekly** period value, or you can specify a time in one of the following formats:

- *HH*:*MM*:*SS* format, where *HH* is the number of hours, *MM* is the number of minutes, and *SS* is the number of seconds, as shown in Table 16. The hours value can be any non-negative integer. The minutes and seconds values must be between 0 and 59.

Table 16. Hours, minutes, and seconds format

72:00:00	=	72 hours
00:30:00	=	30 minutes
00:00:60	=	60 seconds

- *SSSS...* format, where *SSSS...* is a number of seconds, as shown in Table 17. In this format, there is no limit on the seconds value except that it must be a positive integer.

Table 17. Seconds format

60	=	60 seconds
300	=	5 minutes

Defining a periodic schedule

Perform the following steps to define a periodic schedule:

1. Choose the **Define Schedule** command from the **Actions** menu. Enconsole displays the Schedule form.
2. Type the name of the schedule you want to define in the **Name** field, and then select the server or server set to be controlled by the schedule from the option list.
3. Type the initial date and time for the schedule to be started in the **Start** field. When you exit the **Start** field, Enconsole converts the date and time to a standard format (for example, Fri Jan 14, 13:00:00 1996).
4. Choose a period from the Period option list (for example, **weekly**), and then type the number of times to repeat the schedule in the **Repeat** field or choose the repeat value **Forever**.
5. When you have finished defining the schedule, choose the **OK** button on the Schedule form. Enconsole stores the schedule and displays the Command Status display screen. Choose the **OK** button to close the Command Status display screen. The system starts the server as specified by the schedule. Enconsole displays messages on the Serious Messages display screen when the schedule starts each server, and the server state changes to `Running` on the Encina Console display screen.

Using Fast Local Transport (UNIX)

For improved performance, RQS and SFS servers automatically take advantage of the Fast Local Transport (FLT) mechanism when a server and client reside on the same machine. FLT offers faster transport than the DCE RPC.

The following sections describe FLT for servers running under UNIX. On Windows NT, the local RPC mechanism is automatically used when a server and client reside on the same machine.

Using FLT (RQS servers)

Fast Local Transport (FLT) is a method of client/server communication used when a server is local (that is, resides on the same machine as the client). When an RQS server and client reside on the same machine, the RQS

automatically uses the FLT mechanism for many operations. FLT offers faster transport than the DCE RPC. Both client and server must be FLT-enabled. By default, FLT is enabled for RQS clients and servers.

Depending on your operating system, FLT uses either UNIX-domain sockets or named pipes ("FIFO") for pipe-based transport. UNIX-domain sockets and named pipes are automatically created and stored in the **/tmp** directory. The UNIX-domain socket name is **/tmp/EPS***pid* for servers and **/tmp/EPC***pid* for clients. The named pipe files are stored in **/tmp/EFS***pid***/fifo** for servers and **/tmp/EFC***pid***/fifo** for clients.

Normally, a UNIX-domain socket is removed automatically when a process exits. If a process exits abnormally, the socket should be manually removed with the UNIX **rm** command.

The following environment variables control resource usage for pipe-based FLT. In general, we recommend using the default values.

- ENCINA_FLT_PIPE_DIR. The directory used to store UNIX-domain sockets and named pipes (for "FIFO" transport). We recommend that this directory not be on a network file system. The server must have permission to create and remove files in the directory, and clients must have lookup permission to the directory. You may want each server to use a separate directory in which only that server can create and remove files. The default directory is **/tmp**.
- ENCINA_FLT_SERVER_MAX_FDS. This is the maximum number of UNIX file descriptors that the server can use to connect to FLT clients concurrently. Setting this variable to 0 disables FLT connections. The default is 20 less than the file descriptor limit set when the operating system is configured.

  Note: We recommend that the limit on file descriptors be set to 128 for all UNIX platforms. The limit on file descriptors is determined by the UNIX system administrator when a machine is configured; the limit and the method of setting this limit vary by platform. For example, on Solaris systems, the file descriptor limit is 64 by default (at most 44 file descriptors are available for FLT requests; the descriptors unused by FLT are available for other operations, such as communications and file I/O). Because file descriptors are a limited resource, you can set ENCINA_FLT_SERVER_MAX_FDS to a smaller value if you expect the server to handle many operations that require file descriptors (such as accepting many DCE connections via the TCP/IP protocol).
- ENCINA_FLT_CLIENT_MAX_FDS. This controls whether the client can make FLT requests. If the value is set to 0, the client is unable to make FLT

requests. A nonzero value (the default) enables FLT requests. The client can use any number of open file descriptors up to the limit set for the operating system.

- ENCINA_FLT_INACTIVE_TIMEOUT. This is the number of seconds since the server sent a reply to a given client before the server can disconnect the client and reuse its resources (namely, the client's UNIX file descriptor for the pipe-based transport). Smaller timeouts may reduce system performance because of the overhead of reestablishing broken connections. Larger timeouts may prevent active clients from using FLT. The default value is 300 seconds.
- ENCINA_FLT_REPLY_TIMEOUT. This is the maximum number of seconds the server waits for a client's reply pipe to become empty (if the pipe is full when a reply is to be sent). Smaller timeouts may cause the server to discard a reply (resulting in a communication error in the client) under heavy system load. Larger timeouts may tie up threads in the FLT request pool for long periods. The default value is 15 seconds.
- ENCINA_FLT_INITIAL_THREADS. This is the initial number of threads in the FLT request thread pool. The number must be less than the value of ENCINA_FLT_MAX_THREADS. The default value is 1.
- ENCINA_FLT_MAX_THREADS. This is the maximum number of threads in the FLT request thread pool. If all threads are busy when the server receives an FLT request, the server returns the request to the client with instructions to repeat the request via DCE. You can set ENCINA_FLT_MAX_THREADS to match the DCE thread pool size. The default value is 30.

You can disable FLT for a server by setting the environment variable ENCINA_FLT_SERVER_MAX_FDS to 0.

Using FLT (SFS servers)

Fast Local Transport (FLT) is a method of client/server communication used when a server is local (that is, when a server resides on the same machine as the client). When an SFS server and client reside on the same machine, the SFS automatically uses the FLT mechanism for many operations. FLT offers two methods of transport: pipe-based and shared-memory-based. Both offer faster transport than the DCE RPC.

Pipe-based transport uses UNIX-domain sockets or named pipes. Shared-memory-based transport uses shared memory and process-blocking semaphores. Although shared-memory-based transport is faster than pipe-based transport, it offers a lower level of security and imposes some restrictions on the client application.

By default, the pipe-based method is enabled for SFS servers and clients. You must enable the shared-memory-based method by setting an environment variable. When a client and server are on the same machine, SFS first determines whether shared-memory-based FLT is enabled. If shared-memory-based FLT is enabled it is used. If shared-memory-based FLT is disabled or if a shared-memory connection is lost, SFS attempts to use pipe-based transport. If pipe-based transport is disabled, DCE RPC is used. The following sections describe each FLT method.

Using UNIX pipe-based transport

Depending on your operating system, FLT uses either UNIX-domain sockets or named pipes ("FIFO") for pipe-based transport. UNIX-domain sockets and named pipes are automatically created and stored in the **/tmp** directory. The UNIX-domain socket name is **/tmp/EPS***pid* for servers and **/tmp/EPC***pid* for clients. The named pipe files are stored in **/tmp/EFS***pid***/fifo** for servers and**/tmp/EFC***pid***/fifo** for clients.

Normally, a UNIX-domain socket is removed automatically when a process exits. If a process exits abnormally, the socket should be manually removed with the UNIX **rm** command.

By default, the pipe-based FLT method is enabled for SFS servers and clients. The following environment variables control resource usage when pipe-based FLT is being used. In general, we recommend using the default values.

- ENCINA_FLT_PIPE_DIR. The directory used to store UNIX-domain sockets and named pipes (for "FIFO" transport). We recommend that this directory not be on a network file system. The server must have permission to create and remove files in the directory, and clients must have lookup permission to the directory. You may want each server to use a separate directory in which only that server can create and remove files. The default directory is **/tmp**.
- ENCINA_FLT_SERVER_MAX_FDS. This is the maximum number of UNIX file descriptors that the server can use to connect to FLT clients concurrently. Setting this variable to 0 disables FLT connections. The default is 20 less than the file descriptor limit set when the operating system is configured.

  Note: We recommend that the limit on file descriptors be set to 128 for all UNIX platforms. The limit on file descriptors is determined by the UNIX system administrator when a machine is configured; the limit and the method of setting this limit vary by platform. For example, on Solaris systems, the file descriptor limit is 64 by default (at most 44 file descriptors are available for FLT requests; the descriptors unused by FLT are available for other operations, such as communications and file I/O). Because file descriptors are a limited resource, you can set ENCINA_FLT_SERVER_MAX_FDS to a smaller

value if you expect the server to handle many operations that require file descriptors (such as accepting many DCE connections via the TCP/IP protocol).

- ENCINA_FLT_CLIENT_MAX_FDS. This controls whether the client can make FLT requests. If the value is set to 0, the client is unable to make FLT requests. A nonzero value (the default) enables FLT requests. The client can use any number of open file descriptors up to the limit set for the operating system.
- ENCINA_FLT_INACTIVE_TIMEOUT. This is the number of seconds since the server sent a reply to a given client before the server can disconnect the client and reuse its resources (namely, the client's UNIX file descriptor for the pipe-based transport, or its shared memory and semaphore for the shared-memory-based transport). Smaller timeouts may reduce system performance because of the overhead of reestablishing broken connections. Larger timeouts may prevent active clients from using FLT. The default value is 300 seconds. Note that for the shared-memory-based transport a server may disconnect clients at any time if it detects a corruption of the shared memory.
- ENCINA_FLT_REPLY_TIMEOUT. This is the maximum number of seconds the server waits for a client's reply pipe to become empty (if the pipe is full when a reply is to be sent). Smaller timeouts may cause the server to discard a reply (resulting in a communication error in the client) under heavy system load. Larger timeouts may tie up threads in the FLT request pool for long periods. The default value is 15 seconds.
- ENCINA_FLT_INITIAL_THREADS. This is the initial number of threads in the FLT request thread pool. The number must be less than the value of ENCINA_FLT_MAX_THREADS. The default value is 1.
- ENCINA_FLT_MAX_THREADS. This is the maximum number of threads in the FLT request thread pool. If all threads are busy when the server receives an FLT request, the server returns the request to the client with instructions to repeat the request via DCE. You may wish to set ENCINA_FLT_MAX_THREADS to match the DCE thread pool size. The default value is 30.

You can disable pipe-based FLT for a server by setting the environment variable ENCINA_FLT_SERVER_MAX_FDS to 0.

Using shared-memory-based transport

The shared-memory-based transport uses shared memory and process-blocking semaphores. This method is suitable for SFS applications that do not require the SFS client library to be thread-safe. When shared-memory-based transport is used, the entire client process blocks when the client has an SFS request outstanding.

Although shared-memory-based transport offers performance benefits comparable to pipe-based transport, it does not provide the level of security guaranteed by DCE RPC or pipe-based transport. For example, it does not encrypt messages or do any form of authentication. Shared-memory-based transport provides only minimal protection against malicious attacks by using UNIX access control mechanisms on the shared memory segment and semaphores. It does, however, provide reasonable protection against inadvertent scribbling by optionally detaching and reattaching the shared memory segment from the client process when user code is being executed. The environment variables ENCINA_FLT_SMS_PROT_MODE and ENCINA_FLT_SMS_SERVER_OPTIONS control these mechanisms.

Shared-memory-based transport must be enabled on both the client and the server by setting the ENCINA_FLT_SMS_SERVER_ENABLE and ENCINA_FLT_SMS_CLIENT_ENABLE variables to 1. The following environment variables control resource usage for shared-memory-based transport:

- ENCINA_FLT_SMS_SERVER_ENABLE. Enables clients to connect to a server using shared-memory-based transport. The default value is zero, meaning that shared-memory-based transport is disabled. (Set by server.)
- ENCINA_FLT_SMS_CLIENT_ENABLE. Enables servers to connect to a client using shared-memory-based transport. The default value is zero, meaning that shared-memory-based transport is disabled. (Set by client.)
- ENCINA_FLT_SMS_MAX_CONNECTIONS. This is the maximum number of clients that can be connected to an SFS server at one time using shared-memory-based transport. The default value is 16.
- ENCINA_FLT_SMS_MSG_SIZE_MAX. This is the maximum size of messages sent to and received from a server via shared-memory-based transport. The default value is 8192 bytes. In the case of a client communicating with multiple servers using shared memory, the maximum size of the message that can be sent to any of the servers is the smallest value of ENCINA_FLT_SMS_MSG_SIZE_MAX among the servers.
- ENCINA_FLT_INACTIVE_TIMEOUT. This is the number of seconds since the server sent a reply to a given client before the server can disconnect the client and reuse its resources (namely, the client's UNIX file descriptor for the pipe-based transport, or its shared memory and semaphore for the shared-memory-based transport). Smaller timeouts may reduce system performance because of the overhead of reestablishing broken connections. Larger timeouts may prevent active clients from using FLT. The default value is 300 seconds. Note that for the shared-memory-based transport a server may disconnect clients at any time if it detects a corruption of the shared memory.
- ENCINA_FLT_SMS_PROT_MODE. This specifies which processes can access the shared memory segment (using UNIX file-system-style access

mode bits). The only meaningful settings in this context are 666 (allow all processes to read or write the segment); 660 (allow only processes in the same group as the server process to access the segment); and 600 (allow only processes owned by the same user as the server to access the segment). The default value is 660 (octal).

- ENCINA_FLT_SMS_SERVER_OPTIONS. This specifies whether the FLT client detaches shared memory before returning to an application and/or whether a "watch dog" process unblocks clients and deletes shared resources when a server crashes. A value of 1 directs the FLT client to detach shared memory before returning to the application. A value of 2 directs the server to spawn a watch dog process to detect server crashes. When a server crashes, the watch dog process unblocks clients waiting for semaphores and deletes the shared resources owned by the stopped server. A value of 3 specifies that both options are to be used. The default value is 2.

Using Dynamic Resource Manager Management

Encina provides several **tkadmin** commands that enable you to dynamically manage resource manager instances (RMIs). You can use the following commands to enable, disable, list, and query RMIs:

- **tkadmin disable rmi**—This command temporarily disables the flow of XA calls to an RMI.
- **tkadmin enable rmi** —This command reenables an RMI that has been disabled by the **tkadmin disable rmi** command.
- **tkadmin list rmi** —This command lists all RMIs currently registered with a server.
- **tkadmin query rmi** —This command displays information about a particular RMI, including its open and close strings and its current state.

For example, the following command enables the RMI **0** that is registered with the application server named **/.:/branch1/server/merchandise1**:

```
% tkadmin enable rmi -server /.:/branch1/server/merchandise1 0
```

In the following example, the **tkadmin query rmi** command is used to list information about an Oracle RMI (**0**) registered with the application server named **/.:/branch1/server/merchandiseOra**:

```
% tkadmin query rmi  -server /.:/branch1/server/merchandiseOra 0
Information about RMI: 0
Open Info: Oracle_XA+Acc=P/scott/tiger+SesTm=60+LogDir=/tmp+Threads=true
Close Info:
RM name: Oracle_XA
rmFlags: 0
Version: 0
```

```
rmid: 0
Thread Support: 3
tmxaFlags: 1
tmxaState: 0
```

For a complete description of these **tkadmin** commands, refer to *The tkadmin Command Pages*.

Chapter 10. Using the Encina Trace Facility

This chapter introduces the Encina Trace Facility and describes how to enable and redirect trace output with Enconsole. It also describes how to format trace output and dump state information. For information on interpreting trace output and troubleshooting Encina, contact your Encina product support representative.

Introduction to the Encina Trace Facility

The Encina Trace Facility provides tools that monitor error messages, enable selective tracing of execution path events, and dump information about the state of a server. You can use the Encina Trace Facility to specify the type and destination of trace output generated for a server and to manipulate trace output.

This section describes the types of trace output and the tracing that is always enabled. It also describes how to view serious messages with Enconsole and introduces the tools used for tracing.

Types of trace output and destinations

The Encina Trace Facility classifies different types of trace output. Table 18 lists the classes of trace output and their default destinations for Monitor processes.

Table 18. Encina trace classes and default destinations

Trace class	Content of trace output	Default destination (output from cell manager, node manager, and MAS)	Default destination (output from all other Encina resources)
audit	Messages related to security and configuration events	Serious Event Subscription Facility and standard error device (by default, redirected to **server.out**)	Serious Event Subscription Facility
error	Nonfatal error (warning) messages	Serious Event Subscription Facility and standard error device (by default, redirected to **server.out**)	Serious Event Subscription Facility

Table 18. Encina trace classes and default destinations (continued)

Trace class	Content of trace output	Default destination (output from cell manager, node manager, and MAS)	Default destination (output from all other Encina resources)
fatal	Fatal and termination messages	Serious Event Subscription Facility and standard error device (by default, redirected to **server.out**)	Serious Event Subscription Facility
entry	Execution path entry and exit information	None	None
event	Execution path event information	None	None
param	Execution path parameter information	None	None
dump	State information	None	None

All trace output is captured in a main memory ring buffer. You can redirect any class of trace output from the ring buffer to one of the following destinations:

- A file
- A standard input/output (I/O) stream, such as the standard error or standard output devices
- The AIX trace (**trace**) or error logging (**errlog**) facility
- A remote procedure call (RPC) interface for transmitting trace data, such as a Trace Listener server

Fatal, **error**, and **audit** output is always enabled. In the Monitor environment, output is redirected to the Serious Event Subscription Facility and can be monitored via Enconsole and the file **ecm.log**. See "Viewing serious messages" on page 213. For Monitor processes, the output is also sent to the standard error device (by default, **server.out** in the server's working directory). For processes running outside the Monitor, output is directed to the standard error device.

Note: For processes running outside of the Monitor environment on Windows NT systems, output is directed to both the standard error device and the Windows NT Event Log.

By default, **entry**, **event**, **param**, and **dump** output is not enabled, meaning that it is captured only in the main memory ring buffer. The **entry**, **event**, and **param** classes of output can be enabled selectively and redirected by using trace masks; see "Trace specifications" on page 216 for more information. The **dump** class of output can be enabled by using **tkadmin** commands; see "Dumping state information" on page 238 for more information.

Because tracing affects performance, consider the implications of sending trace output to different locations. Sending trace output to a file, I/O stream, AIX facility, or RPC interface can slow performance. Because trace output can be large, use caution when sending trace output to a file or I/O stream.

Viewing serious messages

Serious messages (**fatal**, **error**, and **audit** trace classes) for all servers in a Monitor cell are displayed in Enconsole (as shown in Figure 60).

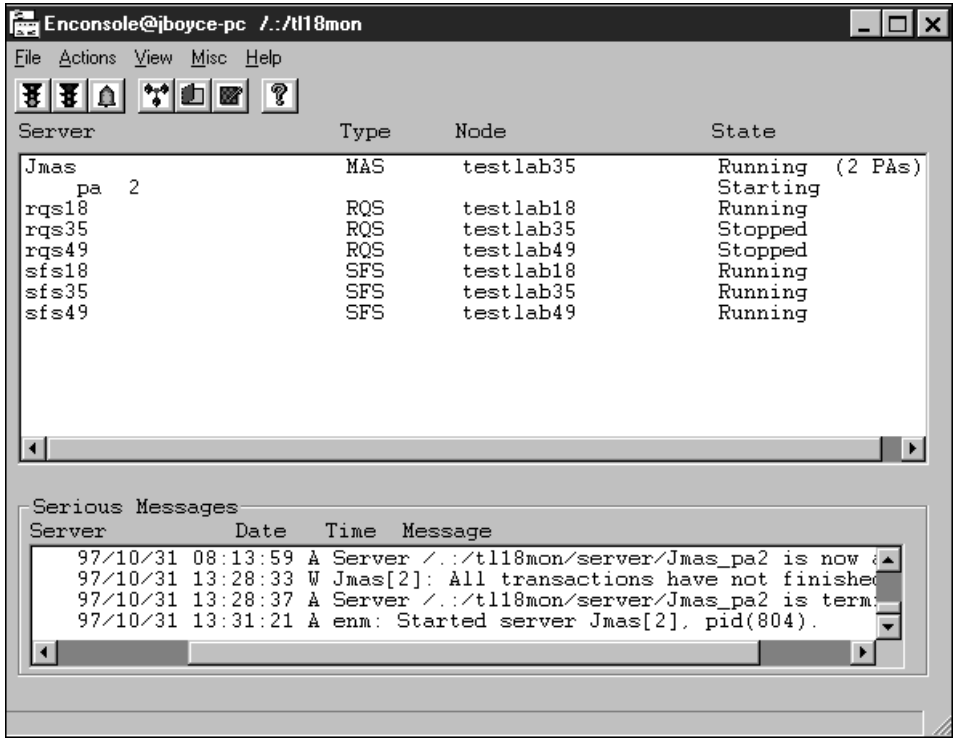

Figure 60. Enconsole serious messages

Enconsole also provides the Serious Messages display screen (shown in Figure 61 on page 214) for viewing and manipulating serious messages. To access the Serious Messages display screen, choose the **Serious Messages** command from the **View** menu. The Serious Messages display screen appears.

The screen is continually updated with new messages. You can prevent the screen from being updated by choosing the **Freeze** button. Choose the **Freeze** button again to enable Enconsole to resume updating the screen.

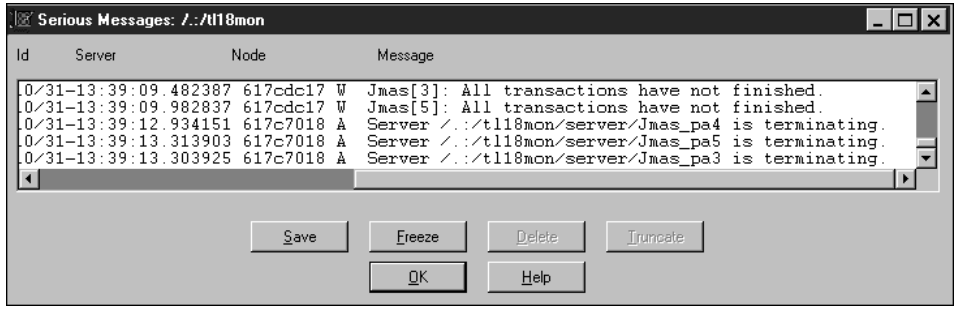

Figure 61. Serious Messages display screen

You can also view serious messages for a specific server by viewing the output file (named **server.out**, by default) in the server's working directory. Use an operating system utility to view the file.

Tools for tracing

Encina provides tools for performing the following trace functions:

- Enabling and redirecting trace output
- Manipulating trace output
- Dumping state information
- Obtaining thread-specific error histories

"How to change tracing" on page 215 describes how to use Enconsole to enable and redirect tracing.

On all platforms, Encina provides the following commands for manipulating trace output. "Manipulating trace output" on page 224 describes how to use each of these commands.

- The **tkadmin dump ringbuffer** command dumps trace output contained in a main memory ring buffer to a file.
- The **tkadmin set ringbuffer size** command changes the size of a ring buffer.
- The **indentTrace** command indents nested functions in trace output to make them more visible.
- The **interpretTrace** command translates a trace binary output file into readable text.
- The **traceListener** command starts a Trace Listener server to accept trace RPC output.

- The **translateError** command translates Encina or DCE status codes.
- The **translateTraceId** command displays information associated with a unique trace identifier.

On Windows NT, WinTrace provides the functionality of the **indentTrace**, **interpretTrace**, **traceListener**, **translateError**, and **translateTraceId** commands in a graphical interface. "Manipulating trace output with WinTrace (Windows NT only)" on page 233 describes how to use WinTrace.

Encina also provides the **tkadmin dump component** and **tkadmin dump product** commands for dumping state information. "Dumping state information" on page 238 describes how to use these commands.

Finally, Encina automatically captures thread-specific error histories in the ring buffer. "Obtaining thread-specific error histories" on page 239 describes the **tkadmin list thread**, **tkadmin query thread**, and **tkadmin name thread** commands used in identifying and viewing error histories.

How to change tracing

This section describes how to use the Encina/DCE Server Options form in Enconsole to specify selective tracing of execution path information (**entry**, **event**, and **param** classes) and trace output destinations. It also describes how to modify a trace mask and redirect trace output. Figure 62 on page 216 provides an example of the Encina/DCE Server Options form, displaying a trace specification.

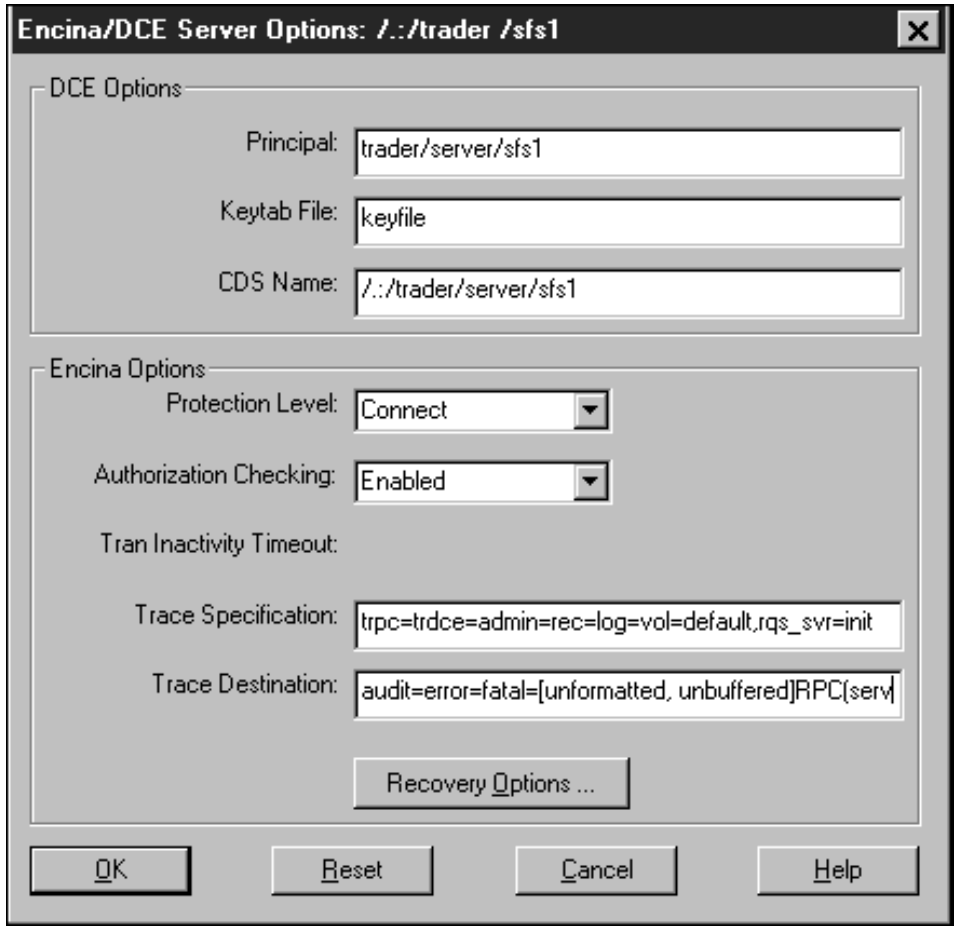

Figure 62. Encina/DCE Server Options form

Note: If you are not using Enconsole, you can control tracing with the **-t** and **-T** options of the server startup command; see the reference page for the startup command for information on using these command-line options. You can also control tracing by using the **tkadmin trace specification** and **tkadmin redirect trace** commands; see the reference pages for these commands for more information.

Trace specifications

You can enable, modify, or disable selective tracing of the **entry**, **event**, and **param** classes of output for a server by using trace masks. A *trace mask* is a 32-bit unsigned integer associated with each Encina product component running in a server. Bits in the trace mask enable various types of tracing in a component. To determine the components running in a server, use the **tkadmin list trace** command; see "Listing components" on page 218.

A *trace option* is used to specify types of tracing and is identified by a string value or its associated hexadecimal value. Bits 0 through 7 (0x000000FF) represent the standard trace options common to all components. The remaining 24 bits represent component-specific trace options. Table 19 lists the standard trace options common to all Encina components. To list the trace options that are specific to a component, use the **tkadmin query trace** command; see "Listing trace options" on page 219.

Table 19. Standard trace options

Trace Option	Hex Value	C Symbolic Constant	Type of Tracing Enabled
trace_event	0x00000001	`TRACE_EVENT`	Events in all functions
trace_entry	0x00000002	`TRACE_ENTRY`	Entry/exit of all exported functions
trace_param	0x00000004	`TRACE_PARAM`	Parameters of all exported functions
trace_export	0x00000006	`TRACE_ENTRY+ TRACE_PARAM`	Entry/exit and parameters of all exported functions
trace_internal_ entry	0x00000008	`TRACE_INTERNAL_ ENTRY`	Entry/exit of nonexported functions
trace_internal_ param	0x00000010	`TRACE_INTERNAL_ PARAM`	Parameters of nonexported functions
trace_global	0x0000001F	`TRACE_GLOBAL`	Entry/exit and parameters of all functions
trace_all, **all**	0xFFFFFFFF	None	All bits in the trace mask

You specify a trace mask by using the **Trace Specification** field of the Encina/DCE Server Options form in Enconsole. The basic syntax of a trace mask specification follows:

```
name=trace_options...
```

You must specify a valid component name for the *name* argument and one or more trace options or their corresponding hexadecimal values for the *trace_options* argument. Use the following syntax conventions when specifying a trace mask:

- To combine trace options by bitwise addition, use the syntax

  ```
  trace_option+trace_option
  ```

- To combine trace options by bitwise subtraction, use the syntax

  trace_option-trace_option
- To set or replace the component's current trace mask with the trace options specified, use the syntax

  name=trace_options...
- To turn off or subtract specific trace options in the current trace mask, use the syntax

  name-=trace_options...
- To add trace options to the current trace mask, use the syntax

  name+=trace_options...
- To disable all tracing for the named component, specify **0** (zero) for the *trace_options* argument.
- To assign the same trace options to multiple component trace masks, use the syntax

  name1=name2=trace_options...

For example, the following trace mask specification enables **entry**, **exit**, and **param** tracing for the **trpc** component:

```
trpc=trace_entry+trace_exit+trace_param
```

The following trace mask specification adds **event** tracing to the current trace mask for the **trpc** component:

```
trpc+=trace_event
```

The following trace mask specification disables **event** tracing for the **trpc** component:

```
trpc-=trace_event
```

The following trace mask specification disables tracing for all components:

```
all=0
```

Listing components

You can list the products and components running in a server and their associated trace masks by using the **tkadmin list trace** command. The command syntax follows:

tkadmin list trace -server *server_name* [**-product** *product_name*] [**-component** *component_name*]

For example, enter the following command to list the products and components running in a server and their associated trace masks:

```
% tkadmin list trace
```

```
Encina Executive:
  epm=0
  admin=default
  tran=trace_entry+trace_param
  trpc=default
  trdce=default
  threadTid=0
Encina BDE:
  bde=0
vendor_bpg:
  vendor_bpg=0
```

As the output indicates, the Encina Executive, Encina Base Development Environment (BDE), and **vendor_bpg** products are listed with their components and associated trace masks. A trace mask value of **0** (zero) indicates that tracing is not enabled for the component. A trace mask value of **default** indicates that the default trace mask for that component is used.

Listing trace options

You can display the current trace mask of a component and all of the trace options available for that component by using the **tkadmin query trace** command. The command syntax follows:

tkadmin query trace -server *server_name component_name*

Specify a component name for the *component_name* argument. For example, enter the following command to display the current trace mask and valid trace options for the **tmxa** component:

```
%  tkadmin query trace tmxa
0x00000401: tmxa_traceMask
0x00000100: xa
0x00000200: lock
0x00000400: callback
0x00000800: xa90
0x00001000: init
```

As the output indicates, the current trace mask value for the component is **0x00000401** or **tmxa_traceMask**. All other valid hexadecimal values and their corresponding string values are also listed.

Trace destinations

By default, **fatal**, **error**, and **audit** trace class output is redirected from the ring buffer to the Serious Event Subscription Facility (where it can be monitored with Enconsole) and the server's standard error device (by default, the **server.out** file in the server's working directory). Messages generated by these classes can greatly impact the operation of the system and should be monitored. These trace classes can be redirected to a different file, but they must be directed to some destination for monitoring (they cannot be allowed to remain in the ring buffer only).

Note: Do not redirect **fatal**, **error**, and **audit** trace messages from any servers managed by the Encina Monitor. Enconsole uses the Serious Event Subscription Facility to display messages on the Encina Console and Serious Messages display screens. If the critical messages are redirected elsewhere, they are not displayed in the Enconsole window.

Specify a trace output destination by using the **Trace Destination** field of the Encina/DCE Server Options form in Enconsole. The basic syntax for a trace output destination follows:

```
trace_class[=trace_class...]=[[option...]]destination_type
[(context_string)]:destination
```

The *trace_class* argument can be one or more of the standard trace classes: **trace**, **fatal**, **error**, **audit**, **dump**, **entry**, **event**, or **param**. Encina also provides the following aliases for frequently used combinations of trace classes:

- The **critical** alias includes the **audit**, **error**, and **fatal** trace classes.
- The **trace** alias includes the **entry**,**event**, and **param** trace classes.
- The **all** alias includes all of the standard trace classes.

Multiple trace classes can be separated by **=** (equal signs).

The *option* argument describes the buffering and formatting characteristics of the output. You can specify any of the following as options: **buffered** or **unbuffered**, and **formatted** or **unformatted**. The buffering and formatting options are independent. The options must be separated by a **,** (comma) and delimited by **[]** (square brackets).

If output is buffered, trace events are stored and sent in batches using a single output operation to reduce performance costs. If output is unbuffered, each trace event is reported in a single output operation. Formatted output is in ASCII format; unformatted output is in binary format. If the specified destination type provides buffering, buffering is used. By default, output is unbuffered and formatted. The **buffered** and **formatted** options can each take an argument. In the form **buffered**=*maximum_size*, the *maximum_size* argument indicates the maximum number of bytes that are buffered to a destination that is temporarily unavailable. The default is 64K. In the form **formatted**=*format_mode*, the *format_mode* argument controls the amount of trace information that is displayed. Its value is an unsigned integer interpreted as a bit mask. If a bit is set in the mask, then the corresponding field appears in the formatted trace output. If no value is specified, all fields except the high-order 16 bits of the process id are displayed. The thread id, prefix, and message always appear in the formatted output. Other fields are controlled by the following bit values:

- 1 — hours, minutes, and seconds
- 2 — date

- 3 — milliseconds
- 4 — microseconds
- 0x10 — low-order 16 bits of process id
- 0x20 — all bits of process id
- 0x40 — use local time instead of GMT
- 0x100 — traceID

The *destination_type* argument directs output to a general destination type. The destination type must appear in uppercase and can be any one of the following:

- The **FILE** destination type refers to a local operating system file. The valid destinations for this class are complete or relative pathnames of a file.
- The **STREAM** destination type refers to well-defined standard I/O streams. The valid destinations are the standard output or standard error device (by default, the **server.out** file in the server's working directory).

  Note: On Windows NT, applications that do not have standard output or standard error devices cannot use the **STREAM** destination type. Such applications can use the environment variable ENCINA_TRACE_REDIRECT to redirect trace output. The variable accepts any valid trace output destination.
- The **AIX** destination type refers to the local tracing and error logging facilities on the AIX operating system. The valid destinations for this type are **trace** and **errlog**.
- The **RPC** destination type refers to an RPC interface for transmitting trace data. The valid objects of this type are DCE RPC binding strings or DCE Directory Service names. This type is intended primarily for redirecting trace events to another program (for example, to a Trace Listener server; see "Starting a trace listener server" on page 231).

  Note: If you write your own server to receive trace redirection RPCs, your server must be able to process multiple trace events in a single RPC. If the trace output is sent unbuffered and unformatted to your server, the trace buffer might contain one or more trace events.
- The **NT** destination type refers to a Windows event logging service. The valid destination for this type is the Windows NT Event Log service (**eventlog**).

The *context_string* argument (an arbitrary string enclosed in parentheses) can be used only with the **RPC** destination type. The context string appears in the output directed to an RPC interface and can be used to distinguish output from different servers writing to the same RPC destination. Conventionally, the context string names the server that is generating the trace output. The

context string must not contain a **)** (right parenthesis) or a **;** (semicolon). If the context string is null, the **()** (parentheses) can be omitted.

The *destination* argument can be one of the following:

- For the **FILE** destination type, a complete or relative pathname of a file. The filename cannot include the special symbols **:** (colon) or **=** (equal sign). If the specified name is a relative pathname, the file is stored relative to the current working directory of the server. If the file already exists, tracing information is appended to it. If the file does not exist, the file is created. By convention, trace output files reside in the primary local directory where the server was started (for example, **/opt/encinalocal/branch1/server/rqs1/trace/events**).
- For the **STREAM** destination type, the standard output or standard error devices of the server.
- For the **AIX** destination type, the **trace** or **errlog** devices. The **trace** device directs trace output to the AIX trace stream, and the **errlog** device directs trace output to the AIX system error log.
- For the **RPC** destination type, a DCE RPC binding string or a DCE Directory Service name.
- For the **NT** destination type, the **eventlog** Event Log service.

For example, the following trace output destination redirects unformatted **event** output to a file named **event.data**:

```
event=[unformatted]FILE:event.data
```

The following trace output destinations redirect **event** output to a Trace Listener server and an RPC string binding, respectively. Note that the server name, **sfs1**, is used as the context string:

```
event=RPC(sfs1):/.:/branch1/server/TraceListener1
event=RPC(sfs1):ncadg_ip_udp:host1[40004]
```

You can list the trace destinations for all trace classes by using the **tkadmin list trace** command. The command syntax follows:

```
tkadmin list redirect -server server_name
```

For example, enter the following command to list the trace destinations of all trace classes for a server:

```
%  tkadmin list redirect
entry:
event:
param:
audit: [formatted,unbuffered]STREAM:stderr
dump:
error: [formatted,unbuffered]STREAM:stderr
fatal: [formatted,unbuffered]STREAM:stderr
```

Modifying trace specifications

You can modify a trace mask when a server is in the `Defined` or `Stopped` state or while it is `Running`. If you modify the trace mask while the server is running, the changes remain in effect only while the server is running and revert to the default settings when the server is stopped and restarted. Otherwise, the changes remain in effect when the server is restarted.

Note: Setting or modifying the trace specification at the command line, by using Enconsole, **enccp**, or **emadmin**overrides the value of the ENCINA_TRACE environment variable.

Perform the following steps to modify a trace mask specification for a server:

1. Choose the **Query/Modify Server** command from the **Actions** menu, and then choose the server from the selection box. Enconsole displays the server definition form for the server.
2. Choose the **Encina/DCE Server Options** button. Enconsole displays the Encina/DCE Server Options form.
3. Click in the **Trace Specification** field and type a new trace mask or edit the current one.
4. Choose the **OK** button on the Encina/DCE Server Options form and then on the server definition form. Enconsole stores the changes, and the Command Status display screen appears.
5. Choose the **OK** button to close the Command Status display screen.

Redirecting trace output

You can redirect tracing when a server is in the `defined` or `stopped` state or while it is `running`. If you redirect tracing while the server is running, the changes remain in effect only while the server is running and revert to the default settings when the server is stopped and restarted. Otherwise, the changes remain in effect when the server is restarted. Perform the following steps to modify the trace output destination for a server:

1. Choose the **Query/Modify Server** command from the **Actions** menu, and then choose the server from the selection box. Enconsole displays the server definition form for the server.
2. Choose the **Encina/DCE Server Options** button. Enconsole displays the Encina/DCE Server Options form.
3. Click in the **Trace Destination** field, and then type a new trace output destination or edit the current one.
4. Choose the **OK** button on the Encina/DCE Server Options form, and then on the server definition form. Enconsole stores the changes, and the Command Status display screen appears.
5. Choose the **OK** button to close the Command Status display screen.

Manipulating trace output

All trace output generated by a server is captured in a main memory ring buffer. The ring buffer is finite in size (the default size is 64K) and can hold only a certain number of the most recent trace output messages, and then the messages are overwritten.

You can manually dump and format the contents of the ring buffer by using the **tkadmin dump ringbuffer** command. You can also have the ring buffer dumped automatically when a traced process encounters certain conditions. This is done by setting Encina environment variables. For more information on manually and automatically dumping the ring buffer, see "Dumping the ring buffer" on page 225. You can also change the size of the ring buffer by using the **tkadmin set ringbuffer size** command. See "Setting the ring buffer size" on page 226 for details.

When the ring buffer is dumped using the methods described in "Dumping the ring buffer" on page 225, by default, a file named **EncinaTraceBuffer.***pid* is created, where *pid* is the process identifier of the server. This file is dumped in binary format. You can translate binary files into readable text by using the **interpretTrace** command (available on all platforms) or WinTrace (on Windows NT). See "Translating trace binary files into readable text" on page 227 for information on using the **interpretTrace** command. See "Manipulating trace output with WinTrace (Windows NT only)" on page 233 for information on using WinTrace.

The default format of trace output is described in "Format of trace output" on page 228.

The Encina Trace Facility contains the following commands for manipulating trace output. These commands are available on all platforms:

- The **indentTrace** command indents trace output to make it more readable. See "Customizing trace output" on page 229 for information on indenting trace output.
- The **translateError** command translates error and status codes into text messages. See "Translating error and status codes" on page 230 for information on translating error and status codes.
- The **translateTraceId** command displays information associated with a trace ID. See "Translating trace identifiers" on page 230 for information on translating trace identifiers.
- The **traceListener** command starts a Trace Listener server, which accepts RPCs containing trace data and sends that data to standard output by default. See "Starting a trace listener server" on page 231 for information on starting a Trace Listener Server.

WinTrace, which is available on Windows NT and Windows 95 only, contains all of the functionality provided by these commands. For more information on using WinTrace, see "Manipulating trace output with WinTrace (Windows NT only)" on page 233.

Dumping the ring buffer

You can manually dump the ring buffer using the **tkadmin dump ringbuffer** command or you can set environment variables to cause an automatic dump of the ring buffer under specific circumstances.

Manual ring buffer dumps

Use the **tkadmin dump ringbuffer** command to dump trace output from the ring buffer. Trace output is sent to the file specified.

Note: Use the **tkadmin dump ringbuffer** command as a diagnostic tool. Using this command during normal server operation can degrade performance.

The complete syntax follows:

```
tkadmin dump ringbuffer -server server_name file_name [-binary] [-overwrite]
```

Specify a filename for the *file_name* argument. By default, the file generated by this command is a readable ASCII file. If the **-binary** option is used, the trace output is in binary format. If the **-overwrite** option is used, the output file is overwritten rather than extended. For example, enter the following command to dump the server's ring buffer into a file named **ringbuffer.out**:

```
% tkadmin dump ringbuffer ringbuffer.out
```

Automatic ring buffer dumps

There are four circumstances under which a ring buffer can be automatically dumped: when a specific trace identifier is encountered, when the ring buffer becomes full, when a process exits, and when a specific signal is received by a process.

To trigger an automatic ring buffer dump when a specific trace identifier (ID) is encountered in a trace event, you must identify the trace ID. To do this, set the value of the ENCINA_TRACE_BUFFER_DUMP_ON_UIDS environment variable to the hexadecimal trace ID. You can identify multiple trace IDs for automatic dump by setting the value of the environment variable to a comma-separated list of the corresponding hexadecimal trace IDs as shown in the following example:

```
setenv ENCINA_TRACE_BUFFER_DUMP_ON_UIDS 280c1825,280c1835,280c1845
```

To trigger a ring buffer dump when the buffer is full, set the ENCINA_TRACE_BUFFER_DUMP_WHEN_FULL environment variable as follows:

```
setenv ENCINA_TRACE_BUFFER_DUMP_WHEN_FULL
```

The ring buffer is dumped before the buffer wraps, and begins to reuse segments.

To trigger a ring buffer dump when a process exits, set the ENCINA_TRACE_BUFFER_DUMP_ON_EXIT environment variable to a nonzero numeric value.

To trigger a ring buffer dump when a specific signal is received, set the ENCINA_TRACE_BUFFER_DUMP_ON_SIGNAL environment variable to the numeric value of that signal. For example, to cause an automatic dump when the signal 31 is received, set the environment variable as follows:

```
setenv ENCINA_TRACE_BUFFER_DUMP_ON_SIGNAL 31
```

When a ring buffer is automatically dumped by using any of these four methods, the contents of the buffer are placed in the file **EncinaTraceBuffer.***pid* (where *pid* is the process ID of the process that caused the automatic dump). You can change the default name of the ring buffer output file by using the ENCINA_TRACE_FILE_FORMAT environment variable. This environment variable takes a **sprintf** string (a string containing %d, where %d defines the position of the *pid* within the string) as shown in the following example:

```
setenv ENCINA_TRACE_FILE_FORMAT MyTraceBuffer.%d
```

Setting the ring buffer size

The size of a ring buffer is set when a server is initialized. By default, the ring buffer is 65536 bytes (64 KB). You can resize the ring buffer in either of the following ways:

- By setting the ENCINA_TRACE_RING_SIZE environment variable. If you use this environment variable, the change to the ring buffer size takes effect the next time the server is started.
- By using the **tkadmin set ringbuffer size** command. If you use this command, the change to the ring buffer size takes effect immediately.

For both the environment variable and the command, specify the size in bytes.

The initial number of segments in a ring buffer is fixed at eight. Each segment is one-eighth of the total size of the ring buffer. You can increase the size of the ring buffer (and thus the number of segments), but you cannot decrease the size below the initial buffer size of 64 KB. The segment size is fixed at 8192 bytes. To make the best use of the available space, the ring buffer size must be a multiple of the segment size.

To display the current size of a ring buffer and its current segment size, use the **tkadmin show ringbuffer size** command. The syntax is as follows:

tkadmin show ringbuffer size -server *server_name*

The following command displays the current ring buffer's total size and its segment size, in bytes, on the standard output stream:

```
%  tkadmin show ringbuffer size
Present Ring Buffer Size:  65536
Present Ring Buffer Segment Size:  8192
```

To set the ENCINA_TRACE_RING_SIZE variable in a C shell, specify the appropriate ring buffer size (for example, **524288** to set the size to 512 KB) as follows:

```
% setenv ENCINA_TRACE_RING_SIZE 524288
```

To set the ring buffer size dynamically, use the **tkadmin set ringbuffer size** command. The syntax is as follows:

tkadmin set ringbuffer size -server *server_name size*

The command sets the ring buffer size and displays its current and new size, as well as its segment size, on the standard output stream. The following example sets the ring buffer size to 85536 bytes and displays the current and new sizes of the ring buffer. Note that the resulting ring buffer size is 90112 (8192 * 11). If you specify a size that is not a multiple of the segment size, the next multiple of the segment size is used.

```
%  tkadmin set ringbuffer size 85536
Present Ring Buffer Size:  65536
Present Ring Buffer Segment Size:  8192
Requested Ring Buffer Size:  85536
Resulting Ring Buffer Size:  90112
```

Translating trace binary files into readable text

Encina provides an interpreter command named **interpretTrace** to translate **EncinaTraceBuffer.***pid* files into readable text format. The complete syntax follows:

interpretTrace [**-r***rep*] [**-m***mode*] [**-s***pid*] [**-f***file*] [**–**] [*files...*]

Specify the process identifier of an **EncinaTraceBuffer.***pid* file for the *pid* argument, a single filename for the *file* argument, or multiple filenames for the *files* argument. Separate each filename with a space. Optionally, you can specify a **0** (zero) for the *mode* argument to make output less verbose. See the reference page for the **interpretTrace** command for more information.

For example, enter the following command to convert the file **EncinaTraceBuffer.17123** to readable text and to send the output to the standard output stream:

```
% interpretTrace -s17123
```

The output from this command can be made more readable by using the **indentTrace** command. See "Customizing trace output" on page 229 for more information.

Format of trace output

Figure 63 provides an example trace message in the default (verbose) format. The thread identifier, process identifier, timestamp, trace identifier, trace class (or subclass) code, and message body fields are labeled.

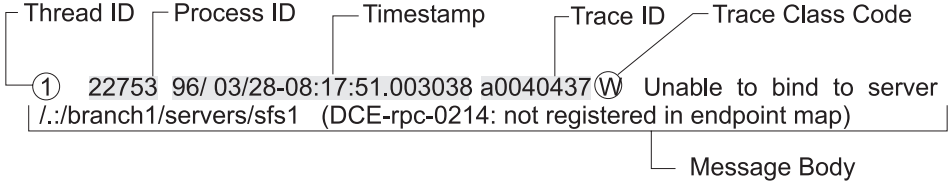

Figure 63. Anatomy of a trace message

A description of each field follows:

- Thread identifier—A system-dependent identifier of the thread in which the event occurred.
- Process identifier—A system-dependent identifier of the process in which the event occurred.
- Timestamp—The date and time at which the event occurred, in hours, minutes, seconds, and microseconds. By default, the timestamp refers to the local time zone.
- Trace identifier—A unique hexadecimal identifier for the message. The trace identifier is provided for customer support purposes or for use when error message catalogs are not available.
- Trace Class Code—A one- or two-character abbreviation representing the class (or subclass) of trace output. This can be one of the following:
  - **>** indicates a function entry
  - **<** indicates a function exit
  - **<R** indicates a function exit with a return value
  - **A** indicates an audit message
  - **D** indicates a state dump message
  - **E** indicates an event message
  - **F** indicates a fatal error message
  - **P** indicates a parameters message
  - **T** indicates a termination message
  - **W** indicates a nonfatal error (warning) message

- Message Body—A description of the event that generated the message. The description can include event-specific data, such as the location of the source file where the event occurred.

The following trace output (in the default format) shows example trace messages for **error** (warning), **event**, **fatal**, **entry**, **param**, **audit**, and **dump** classes:

```
 1    22753 96/01/01-08:17:51.003038 a0040437 W  Unable to bind to
server /.:/branch1/server/sfsServer1 (DCE-rpc-0214: not registered in
endpoint map)
 1    22644 96/01/01-19:40:18.557693 50400415 E  sfs: Hello from
SERVER
31    22644 96/01/01-19:43:43.724026 30349826 F  Undo operation has
repeatedly failed
31    22644 96/01/01-19:43:43.744425 00000006 F  /abc.com/project/
build/src/sfs/server/sfs.c  932
 1    01324 96/01/02-00:27:18.905458 28040c00 >  trpc_UseWkEndpoints
 1    01324 96/01/02-00:27:18.905520 28040c23 P  bindingVectorP:
360d30; count 0.
 1    01324 96/01/02-00:27:18.905840 28040c01 <R trpc_UseWkEndpoints
trpc.c trpc -> 00000000
 1    22719 96/01/03-19:05:45.755217 04800017 A  sfs: Initialized ...
Wed Jan 3 19:05:45 1996
 1    09515 96/01/04-12:35:41.987159 10043819 D  threadTid state
dump for thread id: 1
```

Note that **fatal** (**F**) errors generate two messages. The body of the first message describes the event; the body of the second message includes the name of the source file in which the statement that generated the error appears and the line of the file where the error occurred.

Customizing trace output

Encina provides the **indentTrace** command to accept translated trace output and indent each nested function or event within a function to make it more readable. The complete syntax follows:

indentTrace [**-u**] [*file*]

Specify the name of the file to be used as input for the *file* argument. Optionally, you can specify the **-u** option to cause the output to be unbuffered. For example, enter the following command to indent output of the ASCII file **trace.out**:

```
% indentTrace trace.out
```

The following command uses the **interpretTrace** command and a | (pipe) to format the binary file **EncinaTraceBuffer.17123**, and send it through the **indentTrace** command:

```
% interpretTrace -s17123 | indentTrace
```

Translating error and status codes

Encina provides the **translateError** command to translate Encina or DCE error and status codes into their associated messages. An error or status code can be represented in the following forms in Encina:

- An Encina or DCE status string, such as **ENC-enc-1025** or **DCE-rpc-0036**
- A decimal number, such as **1907401729**
- A hexadecimal number, such as **0x16c9a024**
- An Encina or DCE C constant, such as `TRAN_NOT_READY` or `rpc_s_comm_failure`

The complete syntax follows:

translateError [*codes...*]

Specify one or more status codes to be translated for the *codes* argument. Separate each code with a space. For example, the following command displays the **ENC-tra-1135** status string and its associated message, C constant, and hexadecimal number:

```
% translateError ENC-tra-1135
ENC-tra-1135: All acceptable applications refused to coordinate
[TRAN_ABORT_NO_SUITABLE_COORDINATOR] (0x7796846f)
```

Translating trace identifiers

A *trace identifier* is a unique hexadecimal number associated with a trace message. Encina provides the **translateTraceId** command to display information associated with a trace identifier, including the format string. This information is useful when an error message indicates that there is no format or a bad format associated with a trace identifier, and the message text cannot be displayed. Figure 64 provides an example of a trace identifier.

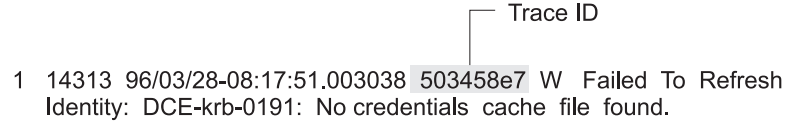

Figure 64. An Encina trace identifier

The complete syntax follows:

translateTraceId [**-fFh**] [*traceIds...*]

Specify one or more trace identifiers to be translated for the *traceIds* argument. You must add the prefix **0x** to each trace identifier to indicate that it is a hexadecimal number. Separate each trace identifier with a space. See the reference page for the **translateTraceId** command for information on using other options with the command.

For example, enter the following command to display the format string associated with the trace identifier **503458e7** and the location in the source code at which the error was detected:

```
%  translateTraceId 0x503458e7

component : sfs_svr
file      : server_sec.c
function  : RefreshIdentity
type      : WARNING
format    : 0
```

Starting a trace listener server

Encina provides a Trace Listener server to accept RPCs containing formatted or binary trace data. The Trace Listener server formats the trace output if necessary and sends the output to its standard output device (by default, the file **server.out**). Trace output must be redirected to the Trace Listener by using Enconsole (see "Redirecting trace output" on page 223) or the **tkadmin redirect trace** command (see the reference page for this command). The Trace Listener accepts trace output from multiple Encina servers; the context string, specified with the trace destination, can be used to identify the source of each trace message.

Use the **traceListener** startup command to start a Trace Listener server. The complete syntax follows:

```
traceListener -n server_name [-h] [-p principal_id [-k keytab_file]]
[-A exclusive_authority] [-Z auth_level] [-N min_authn_level]
[-o option, option...] [-t component:trace_mask[:component:trace_mask...]]
[-T trace_class[=trace_class...]=[[option...]]destination_type:destination
```

Specify the name of the Trace Listener server for the *server_name* argument. You can specify the following options and arguments with this command:

- **-h**—Displays a brief help message about the available options for this command.
- **-p** *principal_id*—Specifies the DCE principal identifier of the server (for example, **branch1/server/listener1**).
- **-k** *keytab_file*—Specifies the keytab file to be used if specifying a value for the *principal_id* argument (for example, **/opt/encinalocal/branch1/server/listener1/keyfile**).
- **-A** *exclusive_authority*—Specifies a principal to be granted exclusive authorization to administer the server and to modify its ACL. This option is required each time a secure server is started so that a principal has access to the server ACL. Permissions granted on ACLs are not checked until exclusive authorization is cleared by using the **tkadmin clear exclusiveauthority** command.

  Before you clear the exclusive authority for a Trace Listener server, you must grant the **administer (a)** permission to an Encina administrative principal (for example, **encina_admin**) so that you can administer the

server. You must also grant the **write (w)** permission to clients of the server so that the clients can redirect trace output to the Trace Listener. Note that the Trace Listener is an ephemeral (nonrecoverable) server. You must recreate the server ACL each time you restart a Trace Listener server.

- **-Z** *auth_level*—Specifies whether authorization checking is enabled in the server. A value of **1** means that authorization checking is enabled and ACLs are checked. The exclusive authority principal has sole authorization to the server. A value of **0** (zero) means that authorization checking is disabled. Once the server is running, authorization checking can be enabled or disabled by using the **tkadmin set authorization** command.
- **-N** *min_authn_level*—Specifies the minimum protection level at which client RPCs to the server are considered to be authenticated. The default value is **2**. See your DCE documentation for more information.
- [**-o** *option, option...*]—Specifies options for the Trace Listener output. Possible arguments for the **-o** option are **format**, **idle=***seconds*, **overwrite**, and **unbuffered**. The **format** argument specifies that the trace output is formatted. The **idle=***seconds* argument specifies that separate trace output files are created, depending on the context of the server that is generating the trace. Modifying the context of the server that is generating trace implicitly changes the output file for the Trace Listener server. The *seconds* variable specifies the number of seconds that can pass without trace information being written to a particular file before it is automatically closed. The **overwrite** argument specifies that existing trace output files are overwritten. The **unbuffered** argument specifies that the trace output is not buffered. Multiple arguments can be used with the **-o** option by using a comma (**,**) to separate the arguments.
- **-t** *component***:***trace_mask*[**:***component***:***trace_mask...*]—Enables tracing of the specified components and trace masks for the Trace Listener server. Note that this trace specification does not affect trace output directed to the Trace Listener server. See "Trace specifications" on page 216 for information about valid trace specifications.
- **-T** *trace_class*[**=***trace_class...*]**=**[[*option...*]]*destination_type***:***destination*—Directs trace output for the Trace Listener server to the specified destination. Note that this trace destination does not affect trace output directed to the Trace Listener server. See "Trace destinations" on page 219 for information about valid trace destinations.

Note: Using the **-t** or **-T** option to enable and direct tracing at the command line overrides the values of the ENCINA_TRACE, ENCINA_TRACE_MASK, and ENCINA_TRACE_SPEC environment variables. To ensure that the correct trace specification is enabled during server initialization in the Monitor environment, consistently defining and modifying trace specifications by using the command line options, the Enconsole interface, **enccp**, or the **emadmin** command suite is recommended.

See the reference page for the **traceListener** command for more information on using these options and arguments.

For example, enter the following command to start a Trace Listener server named **listener1**:

```
% traceListener -n /.:/branch1/server/listener1 -p branch1/server/listener1
-k /opt/encinalocal/branch1/server/listener1/keyfile -A encina_admin
```

Manipulating trace output with WinTrace (Windows NT only)

On Windows NT, you can use individual commands to translate trace output (**interpretTrace**), format trace output (**indentTrace**), translate error and status codes and trace IDs (**translateError** and **translateTraceId**), and start a Trace Listener server (**traceListener**), or you can use WinTrace. WinTrace combines and extends the functionality of the individual commands within a graphical user interface.

WinTrace does *not* provide functionality to manually or automatically dump the trace ring buffer. You must still use the commands and environment variables described in "Dumping the ring buffer" on page 225 to dump the ring buffer.

To open WinTrace, choose WinTrace from the **Start** menu or run the **\opt\encina\bin\winTrace.exe** command. For more information on using WinTrace, see the WinTrace online help.

Opening trace files

In WinTrace, binary and text trace files are opened in the same way. Perform the following steps to open any trace file with WinTrace:

1. Choose the **Open Trace File** command from the **File** menu or choose the **Open** button from the toolbar. The Open file window appears.
2. Specify the pathname of the binary file in the **File Name** field and choose the **OK** button. A new window displays the contents of the file, converted into readable text format. Figure 65 on page 234 provides an example.

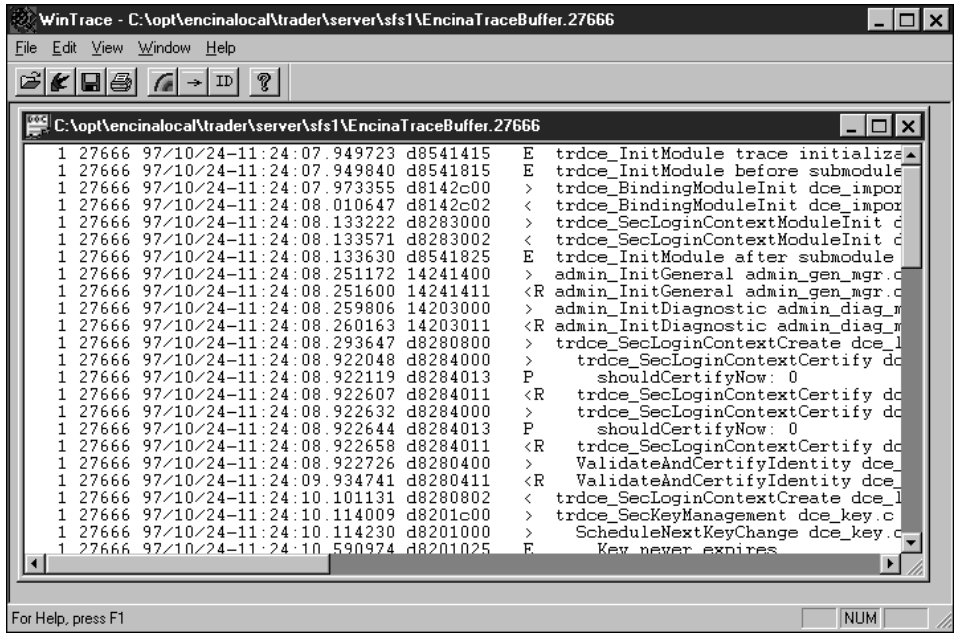

Figure 65. WinTrace

Because WinTrace is fully integrated with the Windows NT environment, you can use any of the standard methods of opening a file. For example, you can

- Drag and drop a file into a WinTrace window
- Associate a file extension with WinTrace so that when you double-click a file with that extension, WinTrace is automatically started and the selected file is opened
- Start WinTrace from the command line and specify a file to open when the program starts

Customizing trace output

You can use WinTrace to indent nested functions within trace output. Perform the following steps to indent trace output:

1. Open a trace output file in WinTrace (see "Translating trace binary files into readable text" on page 227).
2. Position your cursor on the trace output you want to indent and press the right mouse button.
3. Choose the **Indent+** command from the right mouse button menu. The trace output is indented. You can increase the amount of indentation in increments by choosing the **Indent+** command again.

   Note: To decrease or remove indentation, you can choose the **Indent-** command from the right mouse button menu.

Other customization options are available with WinTrace. For example, you can display messages for a specified thread or trace class, or you can specify colors for different types of messages. You can also toggle display of line numbers and verbose format. See the WinTrace online help for information about these features.

Translating error status codes

You can use one of the following methods in WinTrace to translate an error or status code:

- Choose the **Translate Error** button from the toolbar. The Translate Error dialog box appears. Type the error or status code in the **Error Text** field and choose the **Translate** button. A message box displays the translated message. Choose the **OK** button to dismiss the message box.
- Open a trace output file, select the error or status code on the trace output, position your cursor on the selected text, press the right mouse button, and then choose the **Translate Error** command. A message box displays the translated message. Choose the **OK** button to dismiss the message box.

Translating trace identifiers

You can use one of the following methods in WinTrace to translate a trace identifier:

- Choose the **Trace ID** button from the toolbar. The Translate Trace ID dialog box appears. Type the trace identifier in the **Trace ID** field and choose the **Translate** button. A message box displays the translated message. Choose the **OK** button to dismiss the message box.
- Open a trace output file, select the trace identifier on the trace output, position your cursor on the selected text, press the right mouse button, and then choose the **Translate Trace ID** command. A message box displays the translated message. Choose the **OK** button to dismiss the message box.

Starting a trace listener server

You can use WinTrace to configure a Trace Listener server. Perform the following steps to define and start a Trace Listener server:

1. Choose the **Open Trace Listener** command from the **File** menu or choose the **traceListener** button from the toolbar. The TraceListener Options window (shown in Figure 66 on page 236) appears.

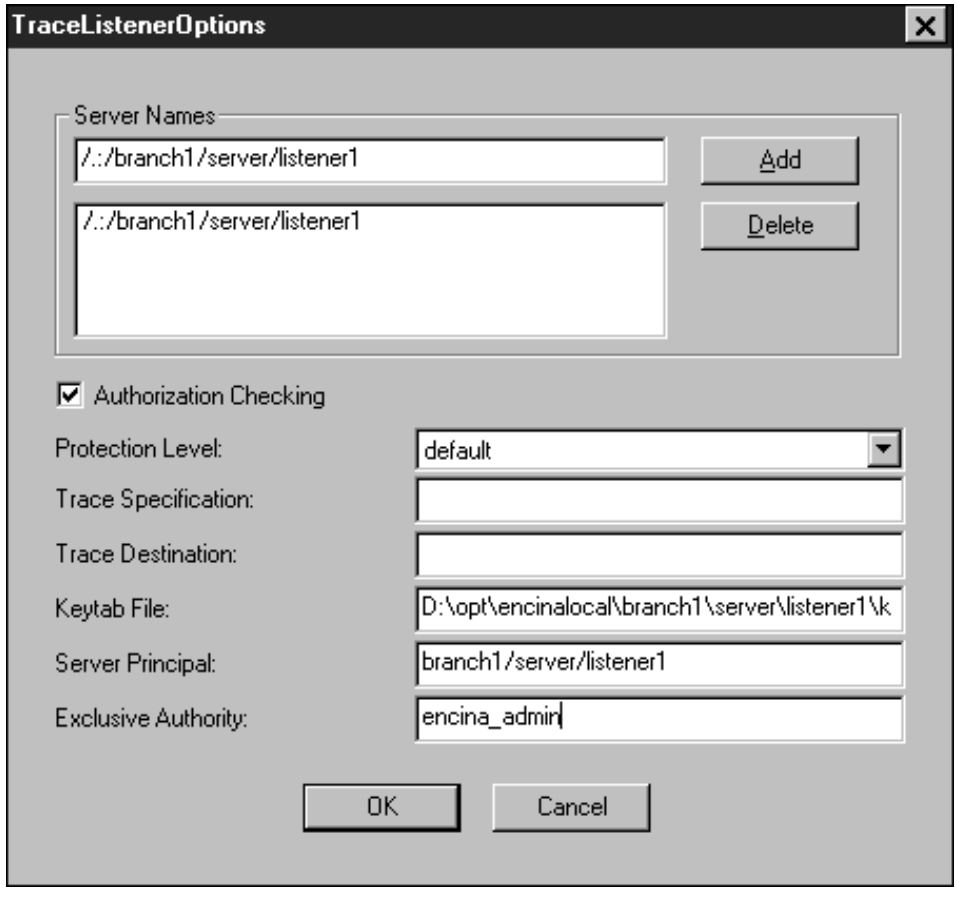

Figure 66. WinTrace TraceListener Options dialog box

2. Type the name of the Trace Listener server in the **Server Names** field (for example, **/.:/branch1/server/listener1**) and choose the **Add** button to add the name to the list of Trace Listener servers. You can delete the server's name from the list by selecting it and then choosing the **Delete** button.
3. Specify whether authorization checking is enabled by selecting the **Authorization Checking** box. When authorization checking is enabled, ACLs are checked. For a server running with DCE security, the exclusive authority principal has sole authorization to the server. Once the server is running, authorization checking can be enabled or disabled with the **tkadmin set authorization** command.
4. Specify a protection level in the **Protection Level** field by typing a value in the field or by selecting a value from the option list. This value specifies the minimum protection level at which client RPCs to the server are considered to be authenticated. The value **default** specifies a protection level of 2. See your DCE documentation for more information.

5. Specify values for the **Trace Specification**, **Trace Destination**, **Keytab File**, **Server Principal**, and **Exclusive Authority** fields.
   - **Trace Specification**—Enables tracing for the Trace Listener server. Note that this trace specification affects tracing for the Trace Listener server itself; it does not affect tracing from other servers that is redirected to the Trace Listener server. See "Trace specifications" on page 216 for information about valid trace specifications.
   - **Trace Destination**—Specifies a trace output destination for the Trace Listener server. Note that this trace destination affects tracing for the Trace Listener server itself; it does not affect tracing from other servers that is redirected to the Trace Listener server. See "Trace destinations" on page 219 for information about valid trace destinations.
   - **Keytab File**—Specifies the full pathname of the keytab file to be used (for example, **D:\opt\encinalocal\branch1\server\listener1\keyfile**). This value is required when starting the server with DCE security.
   - **Server Principal**—Specifies the principal identifier for the Trace Listener server (for example, **branch1/server/listener1**). This value is required when starting the server with DCE security.
   - **Exclusive Authority**—Specifies a principal to be granted exclusive authorization to administer the server and to modify its ACL. This value is required when starting the server with DCE security. Permissions granted on ACLs are not checked until exclusive authorization is cleared by using the **tkadmin clear exclusiveauthority** command. Before you clear the exclusive authority for a Trace Listener server, you must grant the **administer (a)** permission to an Encina administrative principal (for example, **encina_admin**) so that you can administer the server. You must also grant the **write (w)** permission to clients of the server so that the clients can redirect trace output to the Trace Listener. Note that the Trace Listener is an ephemeral (nonrecoverable) server. You must recreate the server ACL each time you restart a Trace Listener server.
6. Choose the **OK** button to start the server and exit the dialog box. A new window appears, displaying the name of the Trace Listener server in the title bar. When the Trace Listener server receives trace RPC output, the output is displayed in the window.

   Note: Once you close the Trace Listener window, the server is stopped. Trace Listener server configurations are not saved. You must reconfigure the server to restart it.

Dumping state information

You can dump the state of an Encina product component (such as **lock**, **log**, or **rec**) or the state of an Encina product (such as the Encina Executive) by using the **tkadmin dump component** and the **tkadmin dump product** commands, respectively. To list the products and components running in a server, use the **tkadmin list trace** command; see "Listing components" on page 218. State information for a component or product includes volatile (in memory) information. The destination for dump output is defined at server startup and can be redirected at any time by using Enconsole or the **tkadmin redirect trace** command.

CAUTION:
Use the tkadmin dump component and tkadmin dump product commands as diagnostic tools. Do not use them during normal server operation.

Dumping the state of a component

Use the **tkadmin dump component** command to dump the state of a component. The complete syntax follows:

```
tkadmin dump component -server server_name component_name
```

Specify a component name for the *component_name* argument. You can determine the components running in a server by using the **tkadmin list trace** command; see "Listing components" on page 218. For example, enter the following command to dump the state of a server's Lock Service (**lock**) component:

```
% tkadmin dump component lock
```

Example dump output for the **lock** component follows:

```
1 D  Lock Service state dump.
1 D  Transaction 65536:
1 D  Holds no locks
1 D  Waiting for no locks
1 D  Transaction 1:
1 D  Holds 5 locks
1 D  Mode: LOCK_MODE_INTENT_READ; Name: index1453; Space: 5.0
1 D  Mode: LOCK_MODE_READ; Name: index1453; Space: 5.0
1 D  Mode: LOCK_MODE_UPGRADE; Name: index666; Space: 5.0
1 D  Mode: LOCK_MODE_WRITE; Name: index9346; Space: 5.0
1 D  Mode: LOCK_MODE_INTENT_WRITE; Name: index13; Space: 0.0
1 D  Waiting for no locks
```

Dumping the state of a product

Use the **tkadmin dump product** command to dump the states of all the components in a product. The complete syntax follows:

```
tkadmin dump product -server server_name product_name
```

Specify a product name for the *product_name* argument. You can determine the products running in a server by using the **tkadmin list trace** command; see "Listing components" on page 218. Product names that contain spaces must be enclosed in double quotation marks (" "). For example, enter the following command to dump the state of all components in the Encina Server Core product:

```
% tkadmin dump product "Encina Server Core"
```

Obtaining thread-specific error histories

The Encina Trace Facility captures error information only for those components that are enabled for tracing. Trace messages are therefore only part of the diagnostic information necessary to resolve problems. Encina automatically captures error histories for threads in all components—not just components that are enabled for trace. The error histories are appended to the ring buffer dump. Error histories provide the following benefits:

- You do not need to know which components are generating errors in order to enable the capturing of those errors. Error histories for threads are independent of any component's trace mask setting. When the Encina trace library encounters a nonzero status code, it is captured, regardless of whether the trace event itself (usually exit trace) is captured.
- You can obtain additional information about generic errors—such as the SFS_COMMUNICATION_ERROR—that result from lower-level errors such as a DCE error. If an application encounters such an error, you can view the most recent errors captured in the error history to determine the underlying cause of the error.
- If you need to determine what lower-level errors caused a transaction to abort in a server, you can extract the error history for each thread.
- You can use error history information in conjunction with trace information to quickly identify which threads have encountered errors. After you have identified specific threads, you can concentrate your investigation on these threads first. Or you can use the error history to identify which components are the best candidates to investigate, and then enable tracing of those components to capture more information.

An error history is created for threads known to the Encina Trace facility—that is, named threads and threads that have returned nonzero status codes. Threads are named either by an explicit call (for example, by a call to the **trace_SetThreadName** function) or when the trace library encounters an error status for the thread. Encina automatically names a thread under any of the following conditions:

- When a thread is created by Encina
- When a thread executes an Encina Transactional Remote Procedure Call (TRPC) (the operation name is used as the thread name)

- When a thread executes administrative commands (the operation name is used as the thread name)

If the thread is not automatically named as described above, or is not explicitly named, the thread becomes known to Encina Trace only when the first error is captured.

An error history displays the last 16 nonzero status codes for a thread. The error history is appended to all ring buffer dumps (the **EncinaTraceBuffer.***pid* file or the file created by the **tkadmin dump ringbuffer** command). A marker separates the error history dump from the ring buffer dump. Error history dumps are grouped by thread, with the newest error code occurring first in each thread's history.

To inspect an error history, use either of the following methods:

- Examine the ringbuffer file.
- Use the **tkadmin list threads** command to find a thread's ID, and then use the **tkadmin query thread** command to display that thread's error history.

The **tkadmin query thread** command has the following syntax:

tkadmin query thread -server *server_name thread_identifier*

The following command displays the current error history for the thread with the identifier **21**:

```
%  tkadmin query thread 21
21  svr_NTOpenFile

  21 14116 95/12/01-11:01:58.713014 50300811 <R svr_NTOpenFile \
server_manager.c sfs_svr -> SFS_OPERATION_TIMED_OUT
  21 14116 95/12/01-11:01:58.712985 50082811 <R svr_OpenFile \
admin_file.c sfs_svr -> SFS_OPERATION_TIMED_OUT
  21 14116 95/12/01-11:01:58.712613 5024e411 <R svr_GetAbortCode \
server.c sfs_svr -> SFS_OPERATION_TIMED_OUT
```

You can assign a name to a thread (that is, replace a thread's numerical ID with a descriptive name). For example, you can associate the name of some logical piece of work with a thread. To name a thread, use the **tkadmin name thread** command. The syntax is as follows:

tkadmin name thread -server *server_name thread_identifier thread_name*

The following command assigns the name **defect87** to the thread identifier **1**:

```
%  tkadmin name thread 1 defect87
```

Note that some threads actively rename themselves; any name assigned to such a thread by using this command is likely to be overwritten.

Part 4. Appendixes

Glossary

A

abort. To fail to commit. When a transaction is aborted, any changes made by the transaction must be undone (rolled back). Except in records in the transaction processing system's log, no evidence of the transaction remains. See also *commit*, *roll back*, *transaction*.

abort complete transaction state. The state in which a transaction has completed all abort-related actions. Abort upcalls and after-resolution callbacks have been completed. The transaction is not yet finished. Additional tasks such as reporting heuristic damage to other participants or completing the requirements of the transaction protocol are not yet complete.

aborted transaction state. The state in which all recovery services have rolled back the work associated with a transaction. No effects of the transaction can be observed by other transactions. After-resolution callbacks take place in this state.

aborting transaction state. The state in which the recovery services are informed that a transaction is aborting.

abstract server stub class. One of the two server stub classes generated from a Transactional Interface Definition Language (TIDL) interface on the server side of an Encina++ client/object application. This class contains virtual functions that map to the remote procedures defined in the server's interface. See also *concrete server stub class*, *stub*, *TIDL*.

accept. PPC: To complete the establishing of communications between two applications using the peer-to-peer communications model. When one application allocates a conversation to another application, the second application accepts that conversation to establish communications with the peer application. See also *allocate*, *conversation*.

access control list. See *ACL*.

access control list entry. See *ACL entry*.

access mode. SFS: A field in the specification of an open file descriptor (OFD) that determines whether other applications can simultaneously access a file. A shared OFD allows more than one application to obtain access to a file at the same time. An exclusive OFD allows only one application to obtain access to a file at a time. See also *exclusive access*, *OFD*, *shared access*.

account. Data in the registry database that allows a principal to authenticate. An account comprises a principal, a primary group, and a primary organization. An account is identified by its principal identifier. See also *authentication*, *principal*, *principal identifier*.

accumulation. RQS: The process of measuring the work associated with a queue. An accumulating queue keeps a running sum of the work as elements are enqueued and dequeued. See also *queue*, *work*.

ACF. Attribute configuration file. An optional companion to an interface definition file (an **.idl** file) that modifies how the Distributed Computing Environment (DCE) Interface Definition Language (IDL) compiler locally interprets the interface definition. See also *IDL*, *TACF*.

ACID properties. The properties of atomicity, consistency, isolation, and durability. These properties ensure that modifications made to data by transactional applications are either completely executed or completely rolled back, are consistent with application requirements, do not conflict, and are permanent in case of system failure. See also *atomicity*, *consistency*, *durability*, *isolation*, *transaction processing system*.

ACL. Access control list. Data that controls access to a protected object. An ACL specifies the

permissions needed to access an object and the permissions that can be granted to principals with the appropriate authentication. See also *ACL entry*, *authentication*, *permission*.

ACL entry. Access control list (ACL) entry. Data in an ACL that specifies a set of permissions. For a principal or group entry, the permissions are those that can be granted to an authenticated principal whose privilege attributes (identity, group membership, or local cell) match those of the entry. For a mask entry (for example, for the **unauthenticated** mask), the permissions are the maximum set that can be granted by a principal or group entry to which the mask applies. See also *ACL*, *permission*, *privilege attribute*.

active index. SFS: A file index updated automatically each time the data in the file is modified. See also *inactive index*, *secondary index*.

active transaction state. The state in which a transaction is currently active in the server. The state indicates that the server has begun but not ended the transaction, the server has received but not replied to a remote procedure call (RPC) for the transaction, or a before-prepare callback is in progress that permits work to be done on the transaction.

active-value attribute. An active-value attribute reflects the value currently in use at the server when the value in use differs from the start-up value in the repository. An active-value attribute is displayed only when its value differs from the value of the corresponding start-up attribute. Active-values attributes end with the **_active** suffix. See also *start-up attribute*.

address. The unique code assigned to each device or workstation connected to a network. A standard Internet address (or IP address) is a 32-bit address field. This field contains two parts. The first part is the network address; the second part is the host number. See also: *IP address* .

administrative group. For Encina, the group whose members have permissions to administer all Encina resources. This group contains the Encina administrative principal. See also *administrative principal*, *group*.

administrative mode. A mode for a server in which all volumes are disabled. When a server is started in this mode, the server exports its administrative remote procedure call (RPC) interfaces only. The server does not perform recovery on its volumes, and its volumes are inaccessible, thus allowing administration but not normal operation. See also *normal operations mode*.

administrative principal. For the Distributed Computing Environment (DCE), the administrative principal (typically **cell_admin**) is the principal who can perform all administrative operations in a DCE cell. For example, this principal can create accounts in the registry database and entries in the Cell Directory Service (CDS) namespace. For Encina, the administrative principal (typically **encina_admin**) is the principal who can perform all administrative operations in an Encina Monitor cell. This principal must be used to define and start servers in the Monitor cell. See also *cold start*, *operator principal*, *principal*.

allocate. PPC: To initiate the establishing of communications between two applications using the peer-to-peer communications model. An application allocates a conversation to request communications with a peer application. See also *accept*, *conversation*, *deallocate*.

allocate tree. PPC: A group of peers in which one peer allocates one or more conversations to other peers, which allocate more conversations, and so forth. See also *allocate*, *conversation*.

alternate record specification. SFS: A way of viewing the data contained in records in an SFS file that is different from the way specified by the record specification used when creating the file.

ancestor. Monitor: An object type that is above a given object type in the type hierarchy. For example, the root type is the ancestor of all object types. The required and optional attributes of an ancestor object type are inherited by descendant object types. An ancestor is also called a supertype. See also *child*, *descendant*, *parent*, *root type*, *type hierarchy*.

API. Application programming interface. Encina provides APIs for the development of distributed transaction processing applications.

application identifier. A unique identifier for an application involved in a transaction in a distributed environment. The application identifier is included in transactional remote procedure calls (TRPCs) sent for that transaction.

application programming interface. See *API*.

application server. The server portion of an Encina application. An application server can be a Monitor application server or a Toolkit server. See also *Monitor application server*, *server*, *Toolkit server*.

atomicity. A basic property of transaction processing systems. Atomicity means that transactions must either complete (be committed) or appear as though they never happened (be aborted). See also *ACID properties*, *transaction*, *transaction processing system*.

attribute. (1) Monitor: A property of an object represented as a named, typed field associated with an object type. (2) SFS: The portion of the information about a file that determines its access and organizational characteristics. The term attribute also refers to a user-defined block of bytes associated with a file.

attribute class. One of the six groups of attributes known to the Monitor. Attributes in a class share characteristics, such as whether the attributes' values can be modified and when their values take effect. See also *active-value attribute*, *dynamic attribute*, *initial start-up attribute*, *read-only attribute*, *runtime attribute*, *start-up attribute*.

attribute list. Monitor: A collection of attribute-value pairs. Each object in the repository of a cell is represented by an attribute list. See also *attribute*.

authentication. The verification of a principal's network identity to obtain network credentials. For interactive principals (users), authentication involves entering a password. For noninteractive principals (server processes), authentication requires a valid key in a keytab file. See also *key*, *keytab file*, *password*, *principal*.

authority. SFS: A field in the specification of an open file descriptor (OFD) identifying the action that can be performed on the file using that OFD. See also *OFD*.

authorization. The determination of a principal's permissions with respect to a protected object. Also, the approval of a privilege sought by a principal with respect to a protected object. See also *permission*, *principal*.

authorization checking. The process of checking permissions for a client's remote procedure calls (RPCs). If authorization checking is enabled, RPCs must have the appropriate Distributed Computing Environment (DCE) credentials on the appropriate access control lists (ACLs). See also *ACL*, *RPC*.

automatic binding. A method of managing the binding for a remote procedure call (RPC) that completely hides binding management from client application code. If the client makes a series of RPCs, the client stub passes the same binding handle with each call. See also *binding handle*, *client stub*, *explicit binding*, *implicit binding*, *transparent binding*.

B

basic conversation. PPC: A conversation that uses a standardized data format as defined by the Systems Network Architecture (SNA). See also *conversation*, *SNA*.

batch call. RQS and SFS: Two or more related but independent functions grouped into a single function call.

before abort transaction state. The state in which the transaction has aborted but recovery services have not been informed that the transaction is aborting. Before-abort callbacks are made in this state. Abort delays leave the transaction in this state after the before-abort callbacks.

binding. A relationship between a client process and a server process involved in a remote procedure call (RPC). A binding requires certain binding information, which includes a communications protocol sequence, a hostname or network address, a server process address (typically an endpoint) on the host, at least one transfer syntax, and an RPC protocol version number. See also *binding file*, *binding handle*, *string binding*.

binding file. An ASCII file that contains one or more entries, each entry consisting of a server name and an associated string binding. The file makes binding information available to clients and servers in the absence of a Directory Service. See also *binding*, *binding handle*.

binding handle. A data object that provides a reference to the binding information for a server. A binding handle allows a client to communicate with a remote server without directly identifying the machine and endpoint at which the server listens for remote procedure calls (RPCs). See also *binding*, *binding file*.

blocking. A situation where data is locked by one transaction so that it cannot be accessed by another. The transaction holding the lock blocks the second transaction's access to the data.

bracket read lock. A shared lock that is obtained, held only for the duration of an operation, and then immediately dropped. This lock mode ensures that only committed data is read. It does not guarantee that successive reads of the data will yield the same value because other transactions can immediately modify the value after the read operation is complete and the lock is dropped.

browser. A client program that initiates requests to a server and displays the returned information.

bucket. RQS: In an RQS server, a repository for one or more element keys in a key table.

C

cache. SFS: An area of memory used to improve performance of some file input/output operations. When a file is cached, some of its records are stored in a temporary buffer and transferred (read or inserted) as a group to reduce the number of remote procedure calls (RPCs).

callback. A function registered by an Encina application to request that an action take place when an event occurs. Unlike upcalls, callbacks are optional actions used internally by applications. Callbacks are associated only with the particular transactions for which they are registered; they are automatically invoked when the transaction events with which they are associated occur. See also *upcall*.

callback retention period. The maximum time, in seconds, that a server retains the registration for a callback. If a server terminates abnormally, the registration entry is lost until the next refresh. See also *callback*.

CDS. Cell Directory Service. The Distributed Computing Environment (DCE) service that manages the naming information used to identify and locate the resources in a DCE cell. CDS is part of the Directory Service. See also *Directory Service*.

CDS directory. Cell Directory Service (CDS) directory. A logical entity for storing entries under a single name in a CDS namespace. A directory can contain object entries for servers, child pointers to other directories, and soft links to other entries. Each physical instance of a CDS directory resides in a different clearinghouse. The instance of a directory that can be modified is called the master replica; all other instances are called read-only replicas. See also *clearinghouse*, *namespace*.

cell. See *DCE cell*, *Monitor cell*.

Cell Directory Service. See *CDS*.

cell manager. Monitor: A Monitor system server that watches and controls all activity within a

Monitor cell. The cell manager communicates with node managers and Distributed Computing Environment (DCE) services. See also *Monitor cell*, *node manager*.

cell-relative name. A name that is meaningful and usable only from within the Distributed Computing Environment (DCE) cell where the entry identified by the name exists. A cell-relative name is a shortened form of a global name. Cell-relative names begin with the cell-relative prefix (**/.:**) and do not contain a cell name. See also *global name*.

certificate. In secure communications, a digital document that binds an encryption key to the identity of the certificate owner so that the certificate owner can be authenticated. A certificate is issued by a certification authority (CA). See also: *encryption* , *certification authority*.

certification authority (CA). In secure communications, a trusted third party (such as VeriSign, Inc.) or a designated internal authority that issues certificates. See also: *certificate* .

certify. To specify a thread as executing on behalf of a transaction in order to prevent the transaction from being recovered while the thread is still executing. See also *decertify*, *thread*.

chain. A series of items, such as structures or actions, in which each item contains some reference to the next item in the series. See also *chaining*.

chaining. A property of transactions that causes another to begin automatically when one ends. See also *chain*.

checkpoint. A snapshot of the current state of the recoverable data being used by an application server. By providing a more recent image of the data, checkpoints of recoverable data made between backups minimize the time required to restart servers that use the data. See also *recoverable data*.

child. Monitor: An object type that is the immediate descendant of another object type (its parent) in the type hierarchy. See also *descendant*, *object type*, *parent*.

chunk. The unit of space allocation that can be used by Toolkit servers. A chunk is larger than a page but smaller than a region. See also *page*, *region*, *volume*.

CICS. Customer Information Control System. CICS is a software package that provides distributed online transaction processing and transaction management.

clearinghouse. A collection of Cell Directory Service (CDS) directories and data that is managed by a CDS Server. A clearinghouse exists as a database file. See also *CDS*, *CDS directory*.

client. A process that accesses the services of a server process. Client programs manage user input, data display, and general output, and they submit requests for data or resources to server programs. (The machine on which a client process runs is sometimes referred to as a client machine.) See also *server*.

client proxy object. An instance of a client stub class created by a client to communicate with a server. A client uses a proxy object to bind to a server object. Each member function call made on the client proxy object then invokes a remote procedure call (RPC) to the server object, which executes the corresponding procedure and returns the result to the client proxy object.

client stub. A code module that is linked with the client portion of an application to translate local procedure calls into remote procedure calls (RPCs). See also *IDL*, *stub*, *TIDL*.

client/object programming model. A model supported by Encina++ classes in which clients access objects instead of servers. Servers export one or more interfaces (classes) and one or more instances of each class (objects). Client applications can access objects exported by servers without regard to how the objects map to servers.

client/server. A model of distributed computing in which applications consist of client processes that request the services and data provided by server processes. Multiple clients typically access the services of a single server. The Distributed

Computing Environment (DCE) and Encina both use the client/server model. See also *client*, *distributed*, *server*.

close string. A string used by an XA-compliant resource manager when the resource manager is closed. See also *open string*.

clustered file. SFS: A file in which records with adjacent key values are physically clustered together.

cold start . The first time a server is started after it has been defined or erased. During a cold start, Encina creates the necessary Distributed Computing Environment (DCE) and Cell Directory Service (CDS) entries and configures and enables the volumes for log and data storage. When a cell manager is cold started, additional necessary groups, principals, and CDS entries are also created. See also *warm start*.

collection period. RQS: A portion of time for which statistics are collected for a queue. The collection period is the elapsed time, in seconds, since queue creation or since the last reset of the collection period. See also *queue*.

COMMAREA. PPC: A buffer that is used to exchange data between a linking program and the linked-to program. See also *linked-to program*, *linking program*.

commit. To make all actions of a transaction permanent. When a transaction commits, all actions associated with it are written to a log. In the event of system problems, the actions are repeated if necessary when the system's recovery mechanism replays the log. See also *abort*, *roll back*, *transaction*.

commit complete transaction state. The state in which a transaction has completed all commitment-related actions. Commit upcalls and after-resolution callbacks have been completed. The transaction is not yet finished. Additional tasks such as reporting heuristic damage to other participants or completing the requirements of the transaction protocol are not yet complete.

commit protocol. A predefined code of conventions that allows multiple processes participating in a distributed transaction to coordinate the committing or aborting of a transaction. See also *recoverable process*, *two-phase commit protocol*.

committed transaction state. The state in which all recovery services have committed the work associated with a transaction such that the results of the work can be observed by other transactions. After-resolution callbacks take place in this state.

committing transaction state. The state in which the recovery services are informed that a transaction has committed.

Common Object Request Broker Architecture. See *CORBA*.

Common Programming Interface–Communications. See *CPI–C*.

Common Programming Interface–Resource Recovery. See *CPI–RR*.

communications area. See *COMMAREA*.

communications manager. See *communications service*.

communications protocol. A clearly defined set of operational rules and procedures for communications.

communications service. A process that enables applications to communicate transactionally with applications that use different communications protocols. PPC Services is a communications service.

component. A software module that is part of an Encina product. For example, the Recovery Service (REC) is a component of the Encina Toolkit. The term component is used interchangeably with module. See also *product*.

concrete server stub class. One of the two server stub classes generated on the server side of an application. This class is derived from the abstract server stub class. Typically, the member functions of the concrete server stub class are

used to implement the remote procedures of an interface. See also *abstract server stub class*, *stub*, *TIDL*.

consistency. A basic property of transaction processing systems. Consistency means that a transaction moves data between consistent states, never leaving data in an inconsistent state. See also *ACID properties*, *transaction*, *transaction processing system*.

consistency constraints. Monitor: Type-specific restrictions on attribute values. Consistency constraints are enforced when objects are created, modified, and destroyed. For example, when a server is added to a server set, the system verifies that the added server exists.

consistency level. SFS: A field in the specification of an open file descriptor (OFD) that determines whether or not changes to data made through the OFD are transactional. See also *OFD*.

context. SFS: The name of the current index, the currently selected key range, and the present location of the current record pointer (CRP) in an SFS file. The context is associated with an open file descriptor (OFD). See also *CRP*, *key range*, *OFD*.

contiguous record. SFS: A record in which all fields are packed into the primary record buffer.

convenience variable . A substitution for known information or command output. Using a convenience variable in interactive commands can reduce typing. In **enccp**, the convenience variable **_ec** (Encina cell) contains the name of the Encina Monitor cell.

conversation. PPC: A mechanism that allows two applications to communicate with one another. A conversation is typically a short-lived logical connection between two transaction programs. All applications that use the Logical Unit (LU) 6.2 protocol communicate with each other via conversations. See also *allocate*, *deallocate*, *LU 6.2*.

coordinator migration. The transfer of the responsibility for resolving a transaction from one participant in that transaction to another. These responsibilities include contacting all of the participants in a transaction, negotiating among them, and managing their actions. See also *transaction coordinator*.

CORBA. Common Object Request Broker Architecture. CORBA provides an infrastructure that allows objects to converse. The infrastructure is independent of the specific platforms and techniques used to implement the objects. Examples of CORBA services are Object Transaction Service (OTS) and Object Concurrency Control Service (OCCS). The services provided by each object serve as the interface between the object and the rest of the system. See also *OCCS*, *OTS*.

CPI–C . Common Programming Interface–Communications. A programming interface specified by IBM Systems Application Architecture (SAA) and X/Open for use in communications between peer applications using the Systems Network Architecture (SNA) Logical Unit (LU) 6.2 protocol. See also *CPI–RR*.

CPI–RR . Common Programming Interface–Resource Recovery. A programming interface specified by IBM Systems Application Architecture (SAA) and X/Open for use in delimiting transactions in a Common Programming Interface–Communications (CPI–C) application. See also *CPI–C*.

critical sections. Portions of shared data to which simultaneous access by multiple threads or applications must be prevented.

CRP. SFS: Current record pointer. A logical indicator used when sequentially processing SFS files or selected ranges of records from those files. The CRP tracks which record in the selected range has just been processed and which is to be processed next. See also *current record*, *key range*, *next record*, *previous record*.

current record. SFS: The record being accessed at present when data is accessed sequentially. See also *CRP*, *next record*, *previous record*.

current record pointer . See *CRP*.

cursor. RQS: A logical client-side object used when examining the objects in a queue sequentially. See also *queue*.

Customer Information Control System. See *CICS*.

customized binding handle. A user-defined data structure from which a primitive binding handle can be derived by user-defined routines in application code. See also *binding*, *binding handle*.

D

Data Definition Language. See *DDL*.

data volume. A logical volume that stores a server's application data. Data volumes are required for cell managers, RQS servers, and SFS servers. See also *log volume*, *logical volume*.

DCE. Distributed Computing Environment. A collection of facilities and services that supports the creation, use, and maintenance of distributed client/server applications in a heterogeneous environment. For Encina, the DCE facilities of interest are the Threads and Remote Procedure Call (RPC) facilities, and the DCE services of interest are the Cell Directory Service (CDS), the Security Service, and the Distributed Time Service (DTS). See also *CDS*, *client/server*, *DCE cell*, *DTS*, *RPC*, *Security Service*, *thread*.

DCE cell. Distributed Computing Environment (DCE) cell. A collection of users, machines, and resources that share common administration. At a minimum, a DCE cell includes one Cell Directory Service (CDS) Server and one Security Server. A DCE cell is the basic unit of DCE operation. Monitor cells are contained within DCE cells. See also *DCE*, *Monitor cell*.

DDL. Data Definition Language. DDL syntax is used to define RQS++ elements and SFS++ records and keys. Each type of record or element is specified as an interface in a DDL file. The DDL file is then compiled using the DDL compiler. See also *RQS++*, *SFS++*.

deadlock. The situation caused by cyclical lock requests that prevent servers from releasing locks needed by other servers. Some servers can detect local deadlocks; however, global deadlocks (those that affect multiple processes) are undetectable and require administrative intervention.

deallocate. PPC: To end a conversation between two applications using the peer-to-peer communications model. Either peer in a conversation can deallocate the conversation. See also *allocate*, *conversation*.

decertify. To specify a thread as no longer executing on behalf of a transaction in order to allow the transaction to be recovered. See also *certify*, *thread*.

default. Monitor: A value assigned to an attribute of a Monitor object type when new objects of that type are created. Defaults can be overridden by explicitly specifying attribute values when an object is created. See also *ancestor*, *child*, *descendant*, *inheritance*, *object type*, *parent*.

DE-Light (DCE Encina Lightweight Client). A set of application programming interfaces (APIs) and a gateway server that are used to enable personal computers and other non-DCE systems to make requests of DCE and Encina servers. See also: *DCE*.

DE-Light security. The protection applied to communications between a DE-Light client and the gateway. DE-Light security uses the Secure Sockets Layer (SSL). See also: *SSL* .

deliver. To cause a callback or an upcall to be invoked. See also *callback*, *upcall*.

dequeue. RQS: To remove the first available element from the head of a queue, subject to locking constraints. The queue can be identified explicitly or it can be selected by the RQS server from a queue set. See also *enqueue*, *requeue*.

descendant. Monitor: An object type that is subordinate to a given object type in the Monitor type hierarchy. For example, all object types are descendants of the root type. Each descendant receives the required and optional attributes of

its ancestor object types. A descendant is also called a subtype. See also *ancestor*, *child*, *object type*, *parent*, *root type*, *type hierarchy*.

Directory Service. The collection of Distributed Computing Environment (DCE) services that makes it possible to access resources anywhere in the network without knowing their physical locations. For example, clients can retrieve the address of a logical name stored in the Cell Directory Service (CDS) to contact the object associated with that name. See also *CDS*.

dirty read. A read operation performed on data that has not been locked by the application performing the read. Consequently, the data being read is not necessarily committed at the time of the read. See also *nolock lock*.

disk. The random-access secondary storage unit identified by a system-specific disk name.

Distinguished Name. In secure communications, the name and address of the person and organization to whom a certificate has been issued. See also: *certificate* .

distributed. Pertaining to the location of programs and services that make up a computing environment on different computer systems but still work together as a single logical entity. See also *client/server*, *DCE*.

Distributed Computing Environment. See *DCE*.

Distributed Program Link. See *DPL*.

Distributed Time Service. See *DTS*.

distributed transaction. A transaction that can update data in multiple user processes on different machines. See also *TRAN*, *transaction*.

Distributed Transaction Service. See *TRAN*.

DLL . Dynamic Link Library. An executable module containing functions that Windows-based programs can call. The module is linked to application programs when they are loaded or run rather than as the final phase of compilation. This means that the same block of library code can be shared among several tasks rather than each task containing copies of the routines it uses. See also *library*.

domain. In an internet, a part of the naming hierarchy. A domain name consists of a sequence of names (labels) separated by periods (dots).

domain name. A name of a host system in a network. A domain name consists of a sequence of names (labels) separated by periods (dots).

domain name server. A server program that supplies address-to-name translation by mapping Internet addresses to domain names. Use of a domain name server allows users to request services of another computer by using a symbolic name, which is easier to remember than an Internet address.

DPL . PPC: Distributed Program Link. An extension to PPC Services that allows Encina applications to communicate with Customer Information Control Systems (CICS) applications by a method that is similar to a transactional remote procedure call (TRPC). Unlike standard PPC applications, a DPL application can act as a client or a server to another Encina DPL or CICS DPL application. See also *CICS*, *mixed-mode LU*, *PPC Services*.

dropping locks. The process of releasing the locks that a transaction holds on data.

DTS. Distributed Time Service. The Distributed Computing Environment (DCE) service that synchronizes the clocks on DCE client and server machines.

duplicate key. SFS: A key in an index whose value is the same as another key index field in that file.

durability. A basic property of transaction processing systems. Durability means that once a transaction commits, any modifications made to data by that transaction are permanent. Transactions that request data modified by a previous transaction always see data that includes all previous modifications. The changes are preserved even in the event of a system

failure. See also *ACID properties*, *transaction*, *transaction processing system*.

dynamic attribute. Monitor: An attribute of a live object that affects the object immediately if the attribute is changed while the object is running. The attribute of a Monitor application server object that represents the number of processing agents is an example of a dynamic attribute. See also *initial start-up attribute*, *live object*, *read-only attribute*, *runtime attribute*, *start-up attribute*.

dynamic endpoint. An address assigned to a server when the server starts. Dynamic endpoints are transient; a server releases its dynamic endpoints when it stops and can be granted different dynamic endpoints when it restarts. Dynamic endpoints are not exported to the namespace. See also *endpoint*, *well-known endpoint*.

Dynamic Link Library. See *DLL*.

E

element. RQS: Typed client data stored in a queue. See also *element ID*, *element key*, *element key table*, *element lifetime*, *element type*, *field*, *queue*.

element ID. RQS: Element identifier. An automatically generated ID associated with an element and unique within an RQS server. An element ID remains associated with an element throughout the element's lifetime. See also *element*.

element key. RQS: A field or a sequence of possibly noncontiguous fields that, taken together, can be used as a basis for retrieving an element. See also *element*.

element key table. RQS: In an RQS server, a table that translates element keys to element identifiers (IDs), permitting efficient retrieval of elements based on keys. Each RQS server has one element key table, which can be administratively tuned. See also *element*.

element lifetime. RQS: The length of time that an element exists in an RQS server, from the time the element is created until the time it is destroyed. See also *element*.

element type. RQS: A named specification that defines the fields of an element and the keys for its retrieval. The formats of RQS elements are compatible with those of SFS records. See also *element*, *field*, *record*.

enccp. Encina Control Program. A command-line and scripting interface for administering Encina Monitor cells. It is modeled after **dcecp**, the Distributed Computing Environment (DCE) administration tool. The Tool command language (TCL) is a portable command language that provides programming facilities, such as variables, procedures, conditionals, list processing functions, and looping constructs. The **enccp** interface extends TCL by providing a set of commands for manipulating Encina Monitor objects. Furthermore, **enccp** incorporates **dcecp** so that all **dcecp** functionality is available from within **enccp**. See also *TCL*.

Encina Control Program. See *enccp*.

Encina server. A server that is provided as part of Encina. RQS servers, SFS servers, and PPC gateway servers are Encina servers. They are also Toolkit servers. See also *Toolkit server*.

Encina-to-Encina conversation. PPC: A conversation between two PPC Executive applications. Encina-to-Encina conversations do not use a gateway server. See also *conversation*, *PPC Services*.

Encina-to-SNA conversation. PPC: Encina to Systems Network Architecture (SNA) conversation. A conversation allocated by a PPC Executive application via a gateway server and accepted by a Logical Unit (LU) 6.2 application running on a mainframe. See also *conversation*, *LU 6.2*, *PPC Services*, *SNA*.

Encina++/CORBA. The Encina++ interfaces that can be used with the Common Object Request Broker Architecture (CORBA) only. These interfaces include client/server support, Tran-C++, the Object Management Group (OMG)

Object Transaction Service (OTS), and the OMG Object Concurrency Control Service (OCCS).

Encina++/CORBA-DCE. The Encina++ interfaces that can be used in a mixed environment that includes both the Common Object Request Broker Architecture (CORBA) and the Distributed Computing Environment (DCE). In this configuration, an application can act as both a DCE and CORBA client or server.

Encina++/DCE. The Encina++ interfaces that can be used in a Distributed Computing Environment (DCE) only. These interfaces include client/server support, Tran-C++, and a subset of the Object Management Group (OMG) Object Transaction Service (OTS).

Enconsole. A graphical user interface (GUI) for centralized, remote administration of a Monitor cell. Enconsole provides a cohesive view of all distributed client/server applications running in a cell, regardless of platform, standard, or protocol. Its display screens can be used to configure and monitor Encina and Distributed Computing Environment (DCE) servers and to review and correct transaction failures. See also *GUI*, *Monitor cell*.

encryption. In secure communications, a means of scrambling data to prevent the data from being read by anyone other than the intended recipient. The sender uses a key to encrypt the message; the recipient uses the decryption key. See also: *key* , *key pair*.

endpoint. The address of a specific server process on a host. For applications that use the Transmission Control Protocol/Internet Protocol (TCP/IP) or the User Datagram Protocol/Internet Protocol (UDP/IP), endpoint numbers are port numbers. See also *dynamic endpoint*, *port*, *well-known endpoint*.

enqueue. RQS: To add an element to the tail of a queue. See also *dequeue*, *requeue*.

entry sequence number. See *ESN*.

entry-sequenced file. SFS: A record-oriented file that stores records in the order in which they are entered into the file. The primary index of an entry-sequenced file is based on the entry sequence numbers (ESNs) corresponding to the order of record insertion. See also *ESN*.

ephemeral process. A process that does not log recoverable data. An ephemeral process is sometimes referred to as an ephemeral application.

ESN. SFS: Entry sequence number. The number that corresponds to the order in which a record is entered into an entry-sequenced file. The primary index of an entry-sequenced file is based on the ESNs of its records. See also *entry-sequenced file*.

events. See *serious messages*.

exclusive access. (1) RQS and SFS: The state of a queue or file during which a single transaction family can access the queue or file without interference. See also *access mode*, *shared access*. (2) Monitor: A restriction imposed by application servers on clients to prevent shared access. Only one client remote procedure call (RPC) can be in progress on a given processing agent at a time.

exclusive lock. A type of lock in which only the transaction holding the lock can access the data in any way. See also *shared lock*, *write lock*.

explicit binding. (1) The method of managing the binding for a remote procedure call (RPC) in which the RPC passes a binding handle as its first parameter. This form of binding gives the client application control over what server machines it contacts. The binding handle is initialized in the application code. See also *automatic binding*, *implicit binding*. (2) Monitor: The binding method in which the issuer of an RPC specifies the application server that receives the RPC. See also *transparent binding*.

export. To make a server interface available to clients. Servers often export binding information for a remote procedure call (RPC) interface to the Cell Directory Service (CDS) namespace, where clients can access it. See also *binding*.

extents. Physically contiguous portions of some physical data storage medium (such as a disk, partition, or tape) that can be individually

addressed by the Log Service. Extents are the basic unit of log data storage.

External File Handler. See *EXTFH*.

EXTFH. External File Handler. A component of Encina that allows COBOL applications to transparently use SFS files for record storage. To the COBOL programmer, there is no apparent difference between EXTFH and standard COBOL input/output; the routines to access data are the same. See also *record*, *SFS*.

F

factory. An object designed to create other objects that are managed by a server. In Encina++, clients bind to a factory object to request the creation of a server object. See also *client/object programming model*.

Fast Local Transport. See *FLT*.

field. RQS and SFS: An individually addressable subdivision of an RQS element or SFS record containing a specific portion of data. For example, a record of data about an employee might be subdivided into fields containing the employee's name, identification number, and salary. See also *element*, *record*.

finished transaction state. The state in which all transaction service functions have taken place. After-finished callbacks take place in this state.

fixed-length field. RQS and SFS: A field in an RQS element or SFS record whose length is always the same. For some fixed-length field types (for example, string types), the amount of data in the field can vary. However, the size of the field itself does not change. See also *variable-length field*.

FLT. RQS and SFS: Fast Local Transport. A method of client/server communication used when a server resides on the same machine as the client. FLT is faster than Distributed Computing Environment (DCE) remote procedure calls (RPCs). Depending on the needs of the application, FLT can be either pipe based or shared-memory based.

fuzzy backup. The backup files created for a volume while the volume is available online. The volume can be changed after the backup files are created. A fuzzy backup can be used as part of the procedure to restore a volume to its latest consistent state.

G

gateway. A functional unit that connects a local data network with another network.

gateway LU. PPC: Gateway Logical Unit (LU). An LU configured in the Systems Network Architecture (SNA) communications package for the machine on which a PPC gateway server is running. The gateway server uses a gateway LU on behalf of a PPC Executive application when communicating with a mainframe. (A gateway LU is sometimes referred to as a local LU.) See also *LU*, *PPC Services*, *SNA*.

global name. A name that is universally meaningful and usable from anywhere in the Distributed Computing Environment (DCE) naming environment. The prefix **/...** indicates that a name is global. See also *cell-relative name*.

global transaction identifier. See *GTID*.

graphical user interface. See *GUI*.

group. A named set of principals who can be granted common permissions via a single access control list (ACL) entry. Groups are represented as entries in the registry database. See also *ACL entry*, *permission*, *principal*.

GTID. Global transaction identifier. An identifier that ties a transaction to all of its participating applications across different servers. The GTID is unique across all servers. See also *TID*.

GUI. Graphical user interface. An interface that uses visual windowing conventions, such as screens and menus, to convey information and provide access to functionality. Encina provides an administrative GUI, Enconsole. See also *Enconsole*.

H

handle. A data object used to access another private data object. For example, a binding handle allows a process to communicate with a remote server without directly identifying the machine and endpoint at which the server listens for remote procedure calls (RPCs). Handles enable modularity and portability. See also *binding*, *binding handle*.

heuristics. Guidelines that a system administrator follows to intervene where a two-phase commit or abort would otherwise fail. When administrators make heuristic decisions, they use their knowledge of an application to force a transaction to commit or abort. See also *two-phase commit protocol*.

hostname. A name, such as zamboni.transarc.com, that is defined for an IP address, such as 158.98.62.45.

HTML (Hypertext Markup Language). A language used to create hypertext documents. Hypertext documents can include links to other related documents. HTML controls the format of text and position of form input areas, for example, as well as the navigable links.

HTML document. A document written in HTML that can contain links to other documents that contain additional information about related terms or subjects.

HTTP (Hypertext Transfer Protocol). The protocol used to transfer and display hypertext documents.

HTTPS (Hypertext Transfer Protocol over the Secure Sockets Layer). The protocol used to transfer and display hypertext documents using SSL.

I

IDL. Interface Definition Language. A high-level language provided by the Distributed Computing Environment (DCE) to facilitate the development of remote procedure call (RPC) interfaces. IDL simplifies the writing of code using RPCs. The IDL compiler is named **idl**. See also *RPC*, *stub*, *TIDL*.

idle timeout. SFS: The time that an open file descriptor (OFD) is permitted to be idle while waiting for an SFS call. After this time period elapses, if an operation using another OFD requests a lock held by the idle OFD, any transaction associated with the idle OFD is aborted, the OFD is closed, and the lock is released. See also *OFD*.

implicit binding. The method of managing the binding for a remote procedure call (RPC) in which a global variable in the client application holds a binding handle that the client stub passes to the RPC runtime. See also *automatic binding*, *client stub*, *explicit binding*, *transparent binding*.

inactive index. SFS: A secondary index that is not updated when the data in a file is modified. See also *active index*, *secondary index*.

inactive transaction state. The state in which a transaction that was once active is no longer active. When a transaction is in this state, it is not legal to make remote procedure calls (RPCs) on behalf of the transaction. It is legal to abort the transaction; however, to commit the transaction, the application must be prepared to commit.

inheritance. Monitor: The process through which object types receive attributes and default values from ancestors. An object type inherits the union of the attributes of its ancestors. Each object type also inherits the defaults of its ancestors. An inherited default can be superseded by associating a default value with a subtype. See also *default*, *type hierarchy*.

initial start-up attribute. A server attribute that cannot be changed once a server is configured—that is, it can be modified only before the server is started for the first time or after an erase operation. See also *active-value attribute*, *dynamic attribute*, *read-only attribute*, *runtime attribute*, *start-up attribute*.

instant duration lock. A lock that behaves as if it were obtained and immediately released. Instant duration locks are typically used to determine whether all of the locks required by a transaction are available before attempting to obtain them. This allows a transaction to avoid holding some locks while waiting for other locks to be released by other transactions.

instant read access. RQS: An access mode that provides an instantaneous snapshot of the current state of a queue. This access mode can show data that has not yet been committed.

intention lock. A lock used to control access to hierarchical resources. Intention locks permit an application to avoid locking a large data object, such as a file of records, when it needs to modify only a portion of the data object, such as one record in the file. An intention lock can be obtained on a file to signify that some records in the file may be changing without blocking access to all records in the file.

interface. For an application programming interface (API), the functions that a server makes available to clients. For an administrative interface, the graphical user interface (GUI) or command-line interface used to manage processes and data. See also *API*, *GUI*, *IDL*, *TIDL*.

Interface Definition Language. See *IDL*.

internal authorization checking. Monitor: The process of checking permissions for remote procedure calls (RPCs) between the cell manager and node managers. If internal authorization checking is enabled, RPCs between objects within the Monitor cell must have the appropriate Distributed Computing Environment (DCE) protection levels.

internal protection level. Monitor: The Distributed Computing Environment (DCE) protection level for remote procedure calls (RPCs) between the cell manager and node managers.

Internet. A wide area network connecting thousands of disparate networks in industry, education, government, and research. The Internet network uses Transmission Control Protocol/Internet Protocol (TCP/IP) as the standard protocol for transmitting information.

IP address (Internet Protocol address). The unique 32-bit address that specifies the actual location of each device or workstation in the Internet. For example, 158.98.62.45 is an IP address.

isolation. A basic property of transaction processing systems. Isolation means that the exchange and modification of information by transactions are synchronized and appear as though multiple simultaneous transactions are actually a series of sequential requests. See also *ACID properties*, *transaction*, *transaction processing system*.

isolation level. SFS: A field in the specification of an open file descriptor (OFD) that specifies the degree to which operations on a file are isolated from other operations on the same file. The isolation level determines the locking policy used. See also *OFD*.

K

key. (1) The password used to authenticate a noninteractive principal such as a server process. All principals' keys are securely stored in their accounts in the registry database. Keys are also referred to as encryption keys because they are used to encrypt and decrypt data; only a process that knows a key can decrypt a value encrypted with that key. See also *authentication*, *keytab file*, *password*. (2) RQS and SFS: The field or fields upon which an index is based. See also *field*.

Key Management Facility. A Security Service facility that enables noninteractive principals to manage their keys. See also *key*, *keytab file*.

key pair. In secure communications, a public key and a private key. These digital keys can be used for digital signatures and encryption. See also: *public key* , *private key* .

key range. SFS: A range of records to be processed sequentially. The range of records is selected by specifying key values that bound the

records to be selected or by specifying an individual key value for which all matching records are selected. See also *key*, *record*.

key ring. In secure communications, a file that contains public keys, private keys, trusted roots, and certificates. See also: *public key* , *private key*, *certificate* .

keytab file. A file that resides on the local disk of a machine to store keys for noninteractive principals, such as server processes, that run on that machine. All principals' keys are also securely stored in their accounts in the registry database. See also *key*.

L

library. A compiled collection of calls and data objects. Libraries are linked with application code at compilation time. See also *DLL*.

linked-to program. PPC: A Distributed Program Link (DPL) program that receives and executes instructions from a DPL application on a remote system (the linking program). The linked-to program acts as a server to the linking program. See also *DPL*, *linking program*.

linking program. PPC: A Distributed Program Link (DPL) program that passes control to a DPL application on a remote system (the linked-to program). The linking program acts as a client to the linked-to program. When the linked-to program finishes executing, control returns to the linking program. See also *DPL*, *linked-to program*.

list attribute. Monitor: An attribute whose value is a list. A list is an ordered collection of elements, where each element can be a string, a number, or another list. Elements in a list are not necessarily unique. An example of a list attribute is the attribute that represents the command-line arguments of a server. See also *set attribute*.

listening. The act of waiting for communications, such as incoming remote procedure calls (RPCs). See also *RPC*.

live object. Monitor: An object that corresponds to an entity that has an active existence outside the repository. Modifying an attribute of a live object changes the behavior of the corresponding active entity. An example of a live object is a server object. See also *dynamic attribute*, *read-only attribute*, *repository*, *runtime attribute*, *start-up attribute*.

load balancing. The process of scheduling client requests among servers to avoid having one server become overloaded while others remain idle.

local cell. The Distributed Computing Environment (DCE) cell to which a principal or a machine belongs. See also *DCE cell*.

LOCK. Lock Service. A component of the Encina Toolkit Server Core that permits synchronization of accesses to data. It enables transactions to lock resources before accessing or modifying them. See also *Toolkit Server Core*.

lock. The mechanism used by a transaction to restrict access by other transactions to data on which it depends. The Encina Toolkit includes the Lock Service (LOCK) to restrict access to data. See also *LOCK*, *Toolkit*.

Lock Service. See *LOCK*.

LOG. Log Service. A component of the Encina Toolkit Server Core that provides efficient and stable storage for recording the actions of programs as they update recoverable data. It ensures that accurate records of transactions are retained across system shutdowns and restarts. See also *logging*, *Toolkit Server Core*.

log archive file. A file that stores backup data for updates to a server's data volume. These updates are stored in the log volume as long as there is sufficient space. Log archive files are automatically generated when media archiving is enabled. See also *media archiving*.

log file. A record-structured sequential file that stores information about updates made to recoverable data. A recovery service uses the information stored in a log file to restore modified data to a consistent state in the event of a system failure. See also *log record*, *recoverable data*, *recovery service*.

log force. An action that causes all pending log records to be written to permanent storage. A log force is usually associated with committing a transaction and ensures that the log records associated with that transaction are actually present in the log. Until a log force is done, these records can be stored only in memory and therefore can be vulnerable to system failures.

log record. A predefined structure into which log data is formatted. Records have a specific size and format, and they contain a certain set of related information. A log record can be identified by its log sequence number (LSN). See also *log file*.

log sequence number. See *LSN*.

Log Service. See *LOG*.

log volume. A logical volume that stores transaction log information for a server. Much of the information stored in a log volume is transient, such as metadata about transactions in progress or information about updates made to the server's data volume. Log volumes are required for Encina recoverable servers. See also *data volume*, *logical volume*.

logging. The process of tracking the updates that a transactional application makes to recoverable data. Logging stores information about updates in a log file, which a recovery service uses to undo or re-create updates in the event of a system failure. See also *LOG*, *log file*, *recoverable data*, *recovery service*.

Logical Unit. See *LU*.

Logical Unit of Work. See *LUW*.

Logical Unit of Work Identifier. See *LUWID*.

logical volume. A logical means of addressing physical data storage. Although it can actually be regions of disk space on multiple physical disks, a logical volume appears to be physically contiguous storage. A logical volume manages physical storage by controlling all access and input/output to its constituent physical volumes. See also *physical volume*, *volume*.

long-term reservation. A processing agent (PA) reservation that lasts until the reservation is explicitly canceled by a client. Long-term reservations can be used only by clients that use explicit binding. See also *reservation*, *short-term reservation*.

LSN. Log sequence number. LSNs are numbers that uniquely identify a record within a log file. They increase monotonically in the order in which the records are written.

LU. PPC: Logical Unit. An abstract representation that identifies a remote entity over a Systems Network Architecture (SNA) network. A conversation occurs between applications at a pair of LUs. PPC Services emulate LUs, allowing PPC Executive applications to participate in conversations across an SNA network. See also *PPC Services*, *SNA*.

LU 6.2. PPC: Logical Unit (LU) 6.2. The Systems Network Architecture (SNA) communications protocol for peer-to-peer communications. The LU 6.2 protocol supports the general data stream supported by SNA. See also *LU*, *mixed-mode LU*, *SNA*.

LU alias. PPC: Logical Unit (LU) alias. An abbreviated name assigned to a complete (fully qualified) LU in the Systems Network Architecture (SNA) communications package. The name must match the LU name configured in PPC Services. See also *LU*, *SNA*, *SNA communications package*.

LUW. PPC: Logical Unit of Work. Another term for a transaction. See also *LU*, *transaction*.

LUWID. PPC: Logical Unit of Work Identifier. A Systems Network Architecture (SNA) global transaction identifier (GTID). See also *LUW*.

M

mainframe LU. PPC: Mainframe Logical Unit (LU). An LU configured in the Systems Network Architecture (SNA) network for a host machine, typically a mainframe. (A mainframe LU is sometimes referred to as a remote or partner LU.) See also *LU*, *SNA*.

managed node. Monitor: A node (machine) whose processes are monitored by a node manager. See also *managed server*, *node manager*.

managed server. Monitor: A process monitored by a node manager. A managed server can be a Monitor application server, an Encina server (such as RQS or SFS), a Distributed Computing Environment (DCE) application server, or a daemon process. See also *managed node*, *node manager*.

manager function. A server function that clients can call via a remote procedure call (RPC). Manager functions are prototyped in a Transactional Interface Definition Language (TIDL) or Interface Definition Language (IDL) file and are located in the server application code. See also *IDL*, *TIDL*.

mapped conversation. PPC: A conversation in which peers exchange arbitrary data records in formats agreed upon by the applications involved. See also *conversation*.

marshaling. The process of bundling the parameters for a remote procedure call (RPC) into a package that is sent to a remote process. See also *RPC*, *unmarshaling*.

media archiving. The process of transparently moving older data from a log file to log archive files to free storage space. Media archiving must be enabled for a server's data and log volumes to be recovered. See also *log archive file*, *log file*.

media failure. A failure of a disk or other storage device that causes loss of data. An example is a disk head crash that makes a disk unavailable or destroys data.

mirroring. The process of maintaining multiple identical copies of the same data to increase availability of that data in the event of a failure. For example, mirroring a logical volume involves associating an identical copy of the backing physical volume with the logical volume. See also *logical volume*, *physical volume*.

mixed-mode LU. A Logical Unit (LU) that uses either the LU 6.2 protocol, the Distributed Program Link (DPL) remote procedure call (RPC) protocol, or both. See also *LU*.

mode. PPC: The desired session characteristics specified by the allocating partner in a PPC conversation. Different sessions between Logical Units (LUs) can have different properties and hence use different mode names. Multiple sessions that share common characteristics can be identified by the same mode name. See also *LU*, *PPC Services*.

module. See *component*.

Monitor. An integrated environment for developing, running, and administering distributed transaction processing applications. The Encina Monitor uses a Monitor cell in which servers are managed by a cell manager and node managers. See also *cell manager*, *Monitor cell*, *node manager*.

Monitor application server. An application server that uses the Monitor application programming interface (API). Monitor application servers consist of processes with one or more processing agents (PAs). See also *application server*, *Monitor*, *PA*, *server*.

Monitor cell. A collection of nodes that are managed by the Monitor and administered as a single unit. A Monitor cell contains one cell manager, one node manager per managed node, and application servers that run on the managed nodes. All managed nodes in a Monitor cell must belong to the same Distributed Computing Environment (DCE) cell. See also *cell manager*, *DCE cell*, *managed node*, *managed server*, *Monitor*, *node manager*.

Monitor cell-relative name. A shortened Monitor cell name that is unique within the Distributed Computing Environment (DCE) cell that contains the Monitor cell (for example, **/.:/foreign-trade**). See also *cell-relative name*.

mutex. A type of mutual exclusion mechanism intended for short-term use with internal data structures (as opposed to recoverable data) within an application. Mutexes are thread dependent: if one thread holds a mutex on some

data, no other thread can access that data. See also *mutual exclusion mechanism*, *thread*.

mutual exclusion mechanism. A means for preventing two separately executing pieces of code from interfering with each other's use of a particular data object. For example, if one thread of an application is executing a function that modifies a shared data structure, the application can use a mutual exclusion mechanism to prevent other threads from reading the data before the modifications are complete. See also *mutex*, *thread*.

N

name server. A machine that provides name resolution for a network. Name servers translate symbolic names assigned to networks and machines into the Internet (IP) addresses used by machines.

name service. Another term for a directory service. See also *CDS*, *Directory Service*.

namespace. The complete set of Cell Directory Service (CDS) entries in a Distributed Computing Environment (DCE) cell. Physically, the namespace exists as a collection of clearinghouses managed by CDS Servers. Logically, the namespace is a hierarchical tree of CDS directories located beneath a DCE cell's root directory. See also *CDS*, *CDS directory*, *clearinghouse*.

native resource manager. A resource manager that interacts with the Monitor using transactional remote procedure calls (TRPCs). Native resource managers can coordinate distributed transactions. See also *resource manager*, *XA-compliant resource manager*.

nested transaction. A transaction begun within the scope of another transaction. A nested transaction is sometimes referred to as a subtransaction. See also *top-level transaction*, *transaction*.

next record. SFS: When data is being accessed sequentially, the record designated by the current record pointer (CRP) as the next record to be processed. See also *CRP*, *current record*, *previous record*.

node. Monitor: A machine in a Monitor cell. See also *Monitor cell*.

node manager. Monitor: A Monitor system server that controls all server activity on a single managed node. The node manager monitors application servers. See also *cell manager*, *managed node*, *Monitor cell*.

nolock lock. A dummy lock mode used when a lock mode must be supplied but locking the data is not actually required. Functions specifying this lock mode can read data even when it is currently locked by other operations or transactions, enabling the performance of dirty reads. See also *dirty read*.

none transaction state. The state in which an application has not directly participated in a transaction. The application has neither begun nor received a remote procedure call (RPC) for the transaction. The application possibly obtained the transaction identifier (TID) by using the transaction relationship functions (parent, descendant, or top ancestor) or from a recovery service upcall. When a transaction is in this state, the application is not permitted to perform work on behalf of the transaction.

nonlive object . An object that does not have an active existence outside the Monitor repository. Server set objects, server schedule objects, resource manager objects, and interface objects are nonlive objects.

normal operations mode. A mode for a server in which all volumes are enabled. When a server is started in this mode, all of the server's remote procedure call (RPC) interfaces are available and application data can be written to the server's volumes. If necessary, the server performs recovery on its volumes. See also *administrative mode*.

O

object. Monitor: Any entity known to the Encina Monitor. It is represented as a named, typed attribute list in the Monitor repository. An object is an instance of an object type. See also *attribute list*.

Object Concurrency Control Service. See *OCCS*.

object reference. A unique identifier for a server object that is returned to a client by a factory object when the factory object creates the associated server object. See also *client proxy object*, *factory*, *server object*.

object request broker. See *ORB*.

Object Transaction Service. See *OTS*.

object type. Monitor: A named class of objects with a set of required attributes, a set of optional attributes, and a set of default values. See also *inheritance*, *type hierarchy*.

OCCS. Object Concurrency Control Service. OCCS is the Common Object Request Broker Architecture services (CORBAservices) Common Object Service used to coordinate access to common resources by multiple processes. The OCCS interface defines C++ classes and functions that enable multiple clients to access shared resources.

OFD. SFS: Open file descriptor. A parameter used to access a file. An OFD defines characteristics of file access, such as the extent to which data in the file can be shared. An application obtains an OFD when it needs to open a file. See also *access mode*.

opaque data structure. A data structure used internally by one functional unit of code but exported for limited external use in another functional unit of code (for example, a binding handle). The external uses must neither access the structure's components nor make assumptions about its constitution.

open file descriptor. See *OFD*.

open string. A string that contains information needed by an XA-compliant resource manager when it is opened. See also *close string*.

open timeout. SFS: The length of time an application waits after requesting an exclusive open on a file currently opened by another application. The application abandons the request when the time elapses.

operation timeout. The length of time an application waits for an operation to complete before abandoning the operation.

operator group. A group (usually **encina_operator_group**) whose members have permissions to perform certain routine administrative tasks within Encina, such as stopping and restarting servers. This group contains the operator principal. See also *administrative group*, *group*, *operator principal*.

operator principal. A principal (usually **encina_operator**) that can perform most day-to-day administrative tasks in Encina. The operator principal has permissions to perform all tasks except defining and modifying objects and starting servers for the first time. This principal typically maintains the cell, stopping and restarting the servers as needed. See also *administrative principal*, *operator group*, *principal*.

optional attribute. Monitor: An attribute that can pertain to objects of a specified type but is not required by them. Optional attributes do not need to be specified when objects are created. See also *required attribute*.

ORB. Object request broker. An ORB manages requests between a client and an object implementation. See also *CORBA*.

orphan element. RQS: An element that is present at an RQS server but does not belong to a queue. An element becomes an orphan when a client application removes it from its current queue, and it remains an orphan until it is requeued. The orphan state is a transitional state that occurs within the scope of a single

transaction. Elements that are still orphaned when the transaction completes are deleted. See also *element*, *queue*.

OTS. Object Transaction Service. OTS is the Common Object Request Broker Architecture services (CORBAservices) Common Object Service used to manage transactions in CORBA applications. The Encina OTS interfaces consist of C++ classes and data types that enable the creation and management of distributed transactions in Encina++ applications. The interfaces offer a choice of two transaction-demarcation models (explicit and implicit) and support the use of C++ exceptions. OTS also provides the ability to write clients in Java. See also *CORBA*.

OTS for Encina++/CORBA. The Object Management Group (OMG) Object Transaction Service (OTS) interface for the Common Object Request Broker Architecture (CORBA) only. This interface does not include client/server support, Tran-C++, or the Object Concurrency Control Service (OCCS).

OTS for Encina++/DCE. The Object Management Group (OMG) Object Transaction Service (OTS) interface for the Distributed Computing Environment (DCE) only. This interface does not include client/server support or Tran-C++.

P

PA. Monitor: Processing agent. A process within an application server that handles remote procedure calls (RPCs) from clients. An application server is made up of one or more PAs; each PA is a separate instance of the server code.

PA scheduling . The policy used by a Monitor application server to determine whether a processing agent (PA) permits shared access by clients. By default, PAs permit only exclusive access to clients. Alternatively, the PA can be multithreaded: many clients can be assigned to the PA, and the PA can handle their requests simultaneously. See also *exclusive access*, *shared access*.

page. The smallest unit of physical storage space that can be addressed by the Toolkit Volume Service. Each page contains 4096 bytes. See also *chunk*, *region*, *volume*.

parent. Monitor: An object type that is the immediate ancestor of another object type (its child) in the type hierarchy. Each object type except the root type has a parent. See also *child*.

participant. An application that takes part in a transaction. An application is considered a participant in a transaction if it either initiates the transaction or receives a request on behalf of the transaction. See also *transaction*.

password. A string that a principal provides to authenticate. Noninteractive principals store their passwords as keys in keytab files. See also *authentication*, *key*, *keytab file*.

peer . PPC: An application that communicates with another application through a conversation over a Systems Network Architecture (SNA) network. During a conversation, the peers can exchange data and do work on each other's behalf. PPC Services enables Encina applications to act as peers to Customer Information Control System (CICS) applications running on mainframes. See also *CICS*, *PPC Services*, *SNA*.

Peer-to-Peer Communications Services. See *PPC Services*.

permanence. See *durability*.

permission. The modes of access to a protected object. The number and meaning of permissions for an object are defined by the access control list (ACL) manager of the object. To perform an operation on an object, a principal must be granted the associated permission by an entry on the ACL of the object. See also *ACL*, *ACL entry*.

persistent data. Data that retains its value across multiple executions of transactional applications, regardless of system failures or restarts.

physical volume. Physical regions on disk that are used to store data. The regions do not need to be contiguous and can reside on different

disks. Physical volumes are said to "back" logical volumes. All physical volumes that back a logical volume must have the same chunk size. See also *chunk*, *logical volume*, *volume*.

ping. To determine the availability of a cell manager, node manager, or server in a Monitor cell. Pinging involves sending a test message and waiting for a response.

playing back. The actions of a recovery service when a transaction processing system using recoverable data is restarted. When started, the recovery service plays back log records for transactions that were prepared but not actually committed. This guarantees that the state of that recoverable data reflects the records maintained by the transaction processing system. See also *recoverable data*, *recovery service*, *transaction*, *transaction processing system*.

port. The actual point of network communication for a given process on a given machine. The term port is often confused with socket, which is a data structure that contains the port number, protocol sequence, and network address. See also *endpoint*, *socket*.

PPC Services. Encina Peer-to-Peer Communications (PPC) Services. An Encina service that comprises two products: the PPC Executive and the PPC Gateway. The PPC Executive is a library that supports peer-to-peer communications using Common Programming Interface–Communications (CPI–C) or Distributed Program Link (DPL) within the Encina environment. The PPC Gateway server process, when used with the PPC Executive, permits a PPC Executive application to communicate with Logical Unit (LU) 6.2 applications over an SNA network. See also *LU 6.2*, *SNA*.

prepare. To agree to commit a transaction. When a participant in a transaction prepares, that participant agrees to make the transaction's changes to recoverable data permanent. See also *prepare phase*, *transaction*.

prepare phase. The first part of a two-phase commit protocol. In the prepare phase of a transaction, each participant informs the coordinator whether it guarantees to commit. A process that cannot guarantee to commit must abort. See also *resolution phase*, *two-phase commit protocol*.

prepared transaction state. The state in which all participants have agreed to commit a transaction. An application participating in the transaction can no longer perform work on behalf of the transaction or unilaterally decide to abort the transaction. The application must accept the instructions of the transaction coordinator (to commit or to abort).

preparing transaction state. The state in which the recovery services in a server are being asked to prepare a transaction. A recovery service is permitted to do work as necessary to prepare, but no other work is permitted on behalf of a transaction.

present transaction state. The state in which a remote procedure call (RPC) is in progress on behalf of a transaction, but no application module has registered to participate. This state is not currently used.

previous record. SFS: When data is being accessed sequentially, the record designated by the current record pointer (CRP) as the previous record in the range. See also *CRP*, *current record*, *next record*.

primary index. SFS: The index of a file defining the physical organization of the records in that file. See also *secondary index*.

primary key. SFS: The key field or fields of a record defining the primary index. See also *secondary key*.

primary storage area. SFS: A storage area for the user data and primary index data of an SFS file. The primary and secondary storage areas of an SFS file are independent. See also *secondary storage area*.

principal. An entity that can communicate securely with another entity. Principals are represented as entries in the registry database and include users, server processes, and

machines. To authenticate, a principal must have an account. See also *account*, *authentication*.

principal identifier. The name used to identify a principal uniquely. Principals are also identified numerically by universal unique identifiers (UUIDs). See also *principal*, *UUID*.

priority class. RQS: A ranking in importance of a group of queues relative to other groups of queues in the same queue set. Priority class 1 is higher than priority class 3 in the same queue set. Zero or more queues can be assigned to a single priority class. Queues at each priority class are considered collectively for dequeueing. See also *queue*, *queue set*, *service level*.

private key. In secure communications, an algorithmic pattern used to digitally sign messages that can be verified with the corresponding public key. A private key is also used to decrypt messages that were encrypted with the corresponding public key. You keep your private key on your own system in a key ring, protected by a password. See also: *encryption* , *public key*, *key ring*.

privilege attribute. An attribute of a principal that can be associated with a set of permissions on an access control list (ACL) entry. Privilege attributes are identity based and include the name, group memberships, and local cell of the principal. See also *ACL entry*, *permission*, *principal*.

processing agent. See *PA*.

product. The high-level software packages that make up Encina. Typically, an Encina product contains several components. For example, the Encina Executive is an Encina Toolkit product with its own components. See also *component*.

protection level. The degree to which network communications are secured. Distributed Computing Environment (DCE) levels of protection range from none, to authentication at the beginning of the remote procedure call (RPC) session, to encrypting all user data transmitted. Typically, higher protection levels are more secure but incur greater performance penalties than lower protection levels.

protocol. A set of rules that describes how to transmit data. Protocols can determine low-level details of machine-to-machine interfaces, such as the order in which bits from a byte are sent; they can also determine high-level exchanges between application programs, such as file transfer.

protocol sequence. A character string that specifies options for network communications protocols. The string consists of three items: the remote procedure call (RPC) protocol for communications, the network address format, and the network transport protocol for communications. See also *binding*.

public key. In secure communications, an algorithmic pattern used to encrypt messages that can be decrypted only with the corresponding private key. A public key is also used to verify messages that were signed with the corresponding private key. A user broadcasts its public key to everyone who will need to exchange encrypted messages with it. See also: *encryption* , *private key*, *key ring*.

Q

queue. RQS: A linear data structure consisting of application-defined elements of heterogeneous types. In an RQS server, elements are added to and removed from queues generally in a first in, first out (FIFO) order. See also *dequeue*, *element*, *enqueue*, *queue body*, *queue set*, *requeue*.

queue body. RQS: A linked list of pages that contains pointers to the physical locations of queue elements stored elsewhere. If elements are small enough, they are stored in the queue body itself to improve element access time. See also *element*, *queue*.

queue set. RQS: A collection of queues. A queue set is a structure that regulates how elements are selected for dequeueing from its member queues. In a queue set, queues are grouped by priority class. See also *priority class*, *queue*, *service level*.

R

random file access. SFS: An access method in which a record is located based on a key value. See also *sequential file access*.

range of records. See *key range*.

read lock. A shared lock that allows multiple transactions to examine a data item concurrently but does not allow changes to the data item. See also *shared lock*.

read-only attribute. Monitor: An attribute that can be read but not modified. An example of a read-only attribute is the attribute representing a server object's current state.

rebinding. The reestablishment of a communications channel for making remote procedure calls (RPCs) after that channel was closed. See also *binding*.

REC. Recovery Service. A component of the Encina Toolkit Server Core that guarantees the consistency of the permanent data used by a distributed transaction service. It uses log records to undo or re-create transactions. See also *recoverable data*, *recovery service*, *Toolkit Server Core*.

record. (1) A predefined structure into which data is formatted. A record has a specific size and format and contains a certain set of related information. (2) SFS: A record in which the information is organized into one or more fields. See also *field*.

recoverable data. Data whose value persists across system shutdowns and failures. In the most common method used to ensure persistence, changes to recoverable data are recorded in a log file, which can always be replayed to return the data to a valid state. See also *log file*, *logging*, *recoverable process*.

recoverable process. A process that uses a recovery service to ensure that it can correctly access and modify persistent data. A recoverable process logs information about transaction state and changes to data. In the event of a failure, a recoverable process uses the logged information to restore any modified data to a consistent state. (A recoverable process is sometimes referred to as a recoverable application.) See also *recoverable data*, *recoverable server*, *recovery service*.

Recoverable Queueing Service. See *RQS*.

recoverable server. A server that uses a recovery service to ensure that it can restore its data to a consistent state in the event of a media failure. Encina RQS, SFS, and PPC gateway servers are examples of recoverable servers. See also *recoverable process*, *recovery service*, *Toolkit server*.

Recovery Service. See *REC*.

recovery service. A service that logs changes to data so that, in the event of a system failure, the data can be restored to a consistent state. The Encina Toolkit includes the Recovery Service (REC) to guarantee the consistency of data used in transactional applications. See also *log file*, *logging*, *playing back*, *REC*, *recoverable process*, *recoverable server*, *Toolkit*.

region. A collection of pages with contiguous addresses on a single disk. The size of a region cannot be smaller than a chunk. See also *chunk*, *page*, *volume*.

registry database. A collection of Security Service information managed by a Security Server. Each Distributed Computing Environment (DCE) cell has its own registry database, which contains information about the principals, groups, organizations, and accounts that exist in that cell. The registry database exists as a collection of database files. See also *Security Service*.

relative file. SFS: A file organized as an array of fixed-length slots. A record can be inserted in the first free slot in the file, at the end of the file, or in a specific slot in the file identified by a relative slot number (RSN). See also *RSN*.

relative slot number. See *RSN*.

remote LU. PPC: Remote Logical Unit (LU). The LU corresponding to the remote site or peer in a conversation. See also *conversation*, *LU*.

remote procedure call. See *RPC*.

repository. Monitor: The persistent, centralized store of objects that make up an Encina Monitor cell. The objects in the Monitor repository store the administrative data pertaining to the cell. There is one object in the repository for each entity in the cell (for example, each server, node, and server set). There is also one object in the repository for each object type. See also *Monitor cell*.

request. The part of a Uniform Resource Locator (URL) that follows the protocol and server hostname. For example, in the URL **http://www.server.com/Public/conf.html**, the request is **/Public/conf.html**.

requeue. RQS: To move an element from one queue to another. See also *dequeue*, *enqueue*.

required attribute. Monitor: An attribute that must be associated with all objects of a given type. Required attributes do not need to be specified explicitly if default values exist. See also *default*, *optional attribute*.

reservation . An arrangement by the Encina Monitor on behalf of a client to maintain a binding to a processing agent (PA) for a certain amount of time. A client can reserve a PA to ensure that it does not have to wait for the PA. See also *long-term reservation*, *short-term reservation*.

resolution phase. The second part of a two-phase commit protocol. In the resolution phase of a transaction, the coordinator collects the responses of the participants. If all participants are prepared to commit, the transaction commits; otherwise, it aborts. See also *prepare phase*, *two-phase commit protocol*.

resource manager. The part of an application that manages data. Application servers access resource managers. See also *native resource manager*, *XA-compliant resource manager*.

restart file. A file that stores the information needed to restart (warm start) a server by using the data from the server's volumes. The restart file is typically created in the server's working directory when the server is started for the first time. See also *warm start*.

reusable OFD. SFS: Reusable open file descriptor (OFD). An OFD that can be used by multiple transactions. The alternative is an OFD that is automatically closed after a single transaction. See also *OFD*.

ring buffer. A fixed-size buffer that holds only a specified amount of the most recent tracing information.

roll back. To undo any modifications performed on behalf of a transaction that does not complete (is aborted). Any changes made by a transaction that is aborted, for whatever reason, must be undone. See also *abort*, *commit*, *transaction*.

root type. Monitor: The object type from which all other object types descend. The root type is the parent of all object types. See also *parent*, *type hierarchy*.

RPC. Remote procedure call. A procedure call executed by a procedure located in a different address space from the calling code. An RPC is analogous to a standard local procedure call. To the caller, an RPC looks the same as a call to a local procedure. See also *TRPC*.

RPC handle. Remote procedure call (RPC) handle. The means by which clients bind to servers. See also *binding*, *binding handle*.

RQS. Recoverable Queueing Service. An Encina service that provides a queueing facility to enable applications to enqueue and dequeue data transactionally. See also *queue*.

RQS++. The C++ programming language interface to RQS. See also *RQS*.

RSN. SFS: Relative slot number. The number of the slot occupied by a record in a relative file. Each record in a relative file contains a field that holds its RSN. See also *record*, *relative file*.

runtime attribute. Monitor: A read-only attribute of a live object that is not stored in the repository. An example of a runtime attribute is the attribute representing a server object's

process identifier (ID). Runtime attributes are meaningful only while the live object is running. See also *live object*.

S

scattered I/O. SFS: Scattered input/output (I/O). A method of performing file input/output that lets applications do vectored reads and writes. Applications can read data from a file into a number of different data structures simultaneously or gather data from different locations to compose the record to be inserted into a file. See also *record*.

secondary index. SFS: An optional index of a file defining an alternative sequence in which the records of the file can be accessed. A secondary index can be active or inactive. An active index is updated each time the data in the file is modified; an inactive index is not. See also *active index*, *inactive index*, *primary index*.

secondary key. SFS: A key field of a record that defines a secondary index. See also *primary key*, *secondary index*.

secondary storage area. SFS: A storage area that contains information about a secondary index of an SFS file. The primary and secondary storage areas of an SFS file are independent. See also *primary storage area*.

Security registry. See *registry database*.

Security Service. The Distributed Computing Environment (DCE) service that manages the security of a DCE cell. Different components of the Security Service ensure that principals are authenticated, that only authorized principals are granted access to data and resources, and that wire transmissions sent via remote procedure calls (RPCs) are sufficiently protected. See also *authentication*, *authorization*, *protection level*.

selection range. See *key range*.

sequence. A Data Definition Language (DDL) data type used to define an array whose size is set at runtime. To define a sequence, the date type and maximum size of the array are required. See also *DDL*.

sequential file access. SFS: The location of a range of records through key values and the subsequent processing of them in an order related to those key values. The index of the file being accessed does not need to be unique. See also *key range*, *random file access*.

serializability. See *isolation*.

serious messages. System messages belonging to the fatal, error, and audit trace classes. By default, tracing of these messages is enabled and output is redirected from the ring buffer to the standard error device. In Enconsole, serious messages are displayed on the Enconsole Servers display screen and are continually updated throughout the Enconsole session. (Serious messages are sometimes referred to as events.) See also *Enconsole*.

server. A process that provides specialized services to client processes. Typically, server programs synchronize and manage access to centralized data or resources, responding to requests from client programs by returning either a system status code or the data or resources the clients request. Servers are usually implemented as processes that wait for and respond to client requests. (The machine on which a server process runs is sometimes referred to as a server machine.) See also *client*.

server object. An instance of a server class that can handle incoming client requests. Server objects must be created either before a server starts listening for remote procedure calls (RPCs) from clients or dynamically via a factory. See also *factory*, *object reference*.

server priority . A rating used by the Monitor to balance requests among application servers that export the same interface. See also *load balancing*.

server schedule. In Enconsole, a procedural plan that specifies start and stop times for a server or server set. A server schedule can be

used to define one-time or periodic events that are to occur automatically. See also *Enconsole*, *server set*.

server set. A collection of servers that can be started and stopped as a unit. A server can be a member of multiple server sets, and it can be started or stopped independently of other servers in any set to which it belongs. In Enconsole, a server schedule can be defined to start or stop the servers in a set at specified times or intervals. See also *Enconsole*, *server schedule*.

server stub. A code module that is linked with the server portion of an application to implement remote procedure calls (RPCs) sent by clients. See also *IDL*, *stub*, *TIDL*.

service level. RQS: An integer associated with a priority class that determines the ratio of dequeueing chances given to the group of queues at that class relative to other priority classes in the same queue set. A service level is represented as an integer from zero to infinity. A zero service level takes a priority class out of service; a nonzero service level distributes dequeueing service among the priority classes according to the ratios of their service levels; an infinite service level represents continual service. See also *priority class*, *queue*, *queue set*.

session. PPC: A long-lived logical connection between two Logical Units (LUs). One conversation at a time takes place over a single session. When a conversation finishes, its session is available for another conversation to use. See also *conversation*, *LU*.

set attribute. Monitor: An attribute that has a set for its value. A set differs from a list in that the order of the elements in a set is not important but uniqueness is. For example, the attribute representing the collection of interfaces exported by a server is a set; because a server can export an interface only once, an interface name cannot be repeated in the set of values for this attribute. See also *list attribute.*

SFS. Structured File Server. An Encina service that provides a record-oriented file system. SFS offers transactional integrity while supporting large numbers of concurrent users and very large files that can span multiple disks.

SFS++. The C++ programming language interface to SFS. See also *SFS*.

shared access. (1) Monitor: Access to an application server by more than one client at the same time. (2) RQS and SFS: The state of a queue or file during which multiple client transactions running under different user processes can access the queue or file simultaneously. Access is subject to the rights and the rules of transactional locking. See also *access mode*, *exclusive access*.

shared lock. A type of lock in which multiple transactions can simultaneously lock a data item for reading. See also *exclusive lock*, *read lock*.

shared memory facility. Monitor: A Monitor facility that allows processing agents (PAs) within a Monitor application server to share transactionally consistent data by gaining concurrent access to named regions of primary memory. This facility enables a higher rate of performance than if an external resource manager were used.

short-term reservation . A processing agent (PA) reservation that lasts for the duration of one transaction or for one remote procedure call (RPC) if the RPC is not transactional. Short-term reservations are acquired implicitly by transparent binding. They can also be acquired explicitly. See also *long-term reservation*, *reservation*.

side information file. PPC: The file that contains site-specific conversation parameters, such as a scheduler entry name field containing the Logical Unit (LU) alias for the other partner in the conversation. The conversation partner for a PPC Executive application is usually defined in the application's side information file. See also *conversation*, *LU alias*, *PPC Services*.

SNA. Systems Network Architecture. A layered architecture that provides common conventions for network communications. SNA specifies formats and protocols (interfaces) to which different products adhere to effect network-wide

communications. See also *LU 6.2*, *PPC Services*, *SNA communications package*.

SNA communications package. PPC: Systems Network Architecture (SNA) communications package. The PPC gateway uses an SNA communications package to access Logical Unit (LU) 6.2 network resources. See also *LU 6.2*, *SNA*.

SNA-to-Encina conversation. PPC: Systems Network Architecture (SNA) to Encina conversation. A conversation allocated by a Logical Unit (LU) 6.2 application at a mainframe in an SNA network to a PPC Executive application through a PPC gateway server. See also *LU 6.2*, *PPC Services*, *SNA*.

socket. The network location at which a server program listens for incoming remote procedure calls (RPCs). A socket contains the network address of the machine on which the socket is located, the local port number of the socket, and the communications protocol (address family) used to communicate with programs listening on that socket. See also *listening*, *port*.

SSL (Secure Sockets Layer). A popular security scheme developed by Netscape Communications Corporation, along with RSA Data Security, Inc. SSL allows the client to authenticate the server and provides for all data and requests to be encrypted. The Uniform Resource Locator (URL) of a secure server protected by SSL begins with **https** (rather than **http**). See also: *authentication* .

start-up attribute. Monitor: An attribute of a live object that, if changed while the associated entity is running, affects the associated entity the next time the object is started but not immediately. An example of a start-up attribute is the attribute representing a server object's command-line arguments. See also *active-value attribute*, *dynamic attribute*, *read-only attribute*, *runtime attribute*.

start-up dependency. A method of controlling the order in which servers are to be started. Specifying dependencies for a server allows for verification that the resources on which a server depends are made available before the server that depends on them is started.

starvation. RQS: A condition that occurs when queues at lower priority classes are deprived of dequeueing chances because queues at higher priority classes receive continual service. As the service level of a priority class approaches infinity, priority classes with lower service levels risk starvation. See also *queue*, *strict prioritization*, *weighted prioritization*.

strict prioritization. RQS: A queue set configuration in which dequeueing takes place strictly in the order of priority classes. In a queue set with strict prioritization, all priority classes have an infinite service level. Queues at the highest priority class receive continual service, subject to the availability of elements for dequeueing. Queues at the next highest priority class are considered next, and so forth. See also *queue*, *starvation*, *weighted prioritization*.

string binding. A character string used by a client to make a remote procedure call (RPC) to a server. A string binding consists of a communications protocol sequence, a hostname or network address, and a server process address (typically an endpoint or port) on the host. See also *binding*, *binding handle*.

structured file. SFS: A file with data organized into a specific format that is usually record-oriented. See also *record*.

Structured File Server. See *SFS*.

structured file system. SFS: The collection of data managed by a single Structured File Server (SFS). All access to a structured file system is through a single server, using a special type of open file descriptor (OFD). See also *OFD*, *SFS*.

stub. A code module that translates local procedure calls into remote procedure calls (RPCs) for an RPC interface. For Distributed Computing Environment (DCE) applications, stubs are generated by the Interface Definition Language (IDL) compiler, **idl**; for Encina applications, stubs are generated by the Transactional Interface Definition Language

(TIDL) compiler, **tidl**. Both compilers generate client and server stub files, which are linked with the client and server portions of an application during compilation. See also *client stub*, *IDL*, *server stub*, *TIDL*.

subtransaction. See *nested transaction*.

subtype. See *descendant*.

supertype. See *ancestor*.

synchronization level . See *synclevel*.

synclevel. PPC: The level of synchronization for a conversation. The Logical Unit (LU) 6.2 protocol supports three levels of synchronization: none, confirm, and syncpoint. See also *LU 6.2*, *SNA*, *synclevel confirm*, *synclevel none*, *synclevel syncpoint*.

synclevel confirm. PPC: The level of conversation synchronization supported by the Logical Unit (LU) 6.2 protocol that provides a single message acknowledgment rather than full two-phase commit processing. Synclevel confirm is used for nontransactional conversations. It is also called synclevel 1 (SL1). See also *LU 6.2*, *SNA*, *synclevel*.

synclevel none. PPC: The lowest level of conversation synchronization supported by the Logical Unit (LU) 6.2 protocol. Synclevel none provides no built-in synchronization. Synclevel none is used for nontransactional conversations. It is also called synclevel 0 (SL0). See also *LU 6.2*, *SNA*, *synclevel*.

synclevel syncpoint. PPC: The highest level of conversation synchronization supported by the Logical Unit (LU) 6.2 protocol. Synclevel syncpoint is required for transactional conversations. Synclevel syncpoint is the term used by Systems Network Architecture (SNA) for two-phase commit processing. It is also called synclevel 2 (SL2). See also *LU 6.2*, *SNA*, *synclevel*, *two-phase commit protocol*.

syncpoint. PPC: A reference point to which resources can be returned if a failure occurs. One of the peers in a peer-to-peer conversation must call for a syncpoint to commit a Logical Unit of Work (LUW). See also *LUW*, *synclevel syncpoint*.

system exception. An exception thrown by the underlying system rather than by the user application. For example, a system exception can be generated by the remote procedure call (RPC) facility. User applications can catch both system and user exceptions. See also *user exception*.

Systems Network Architecture. See *SNA*.

T

TACF. Transactional attribute configuration file. A TACF is the transactional equivalent of a Distributed Computing Environment (DCE) attribute configuration file (ACF). Like an ACF, a TACF modifies how the Transactional Interface Definition Language (TIDL) compiler interprets the interface definition. See also *ACF*, *TIDL*.

TCL . Tool command language. A portable command language that provides programming facilities, such as variables, procedures, conditionals, list-processing functions, and looping constructs. The **enccp** interface incorporates the scripting capabilities of TCL and can therefore be used to automate Encina administrative tasks. See also *enccp*.

Telshop application. A transactional order entry application delivered with Encina products as a demonstration application. The Telshop application consists of a merchandise application server that manages inventory data in an external database and a client that communicates with the server.

thread. A path of execution (thread of control) within a process. Threads differ from standard processes in that they share access to a common address space rather than the multiple address spaces required by separate processes.

Thread-to-Tid Mapping Service. See *ThreadTid*.

threaded application. An application that performs its function by simultaneously using multiple execution paths (threads of control) within a single address space. See also *thread*.

ThreadTid. Thread-to-Tid Mapping Service. A component of the Encina Toolkit Executive that maintains the association between a thread and a transaction identifier (TID). It allows applications to determine which transaction is associated with a particular thread. See also *TID*, *Toolkit Executive*.

TID. Transaction identifier. A unique ID assigned to each transaction to identify all actions associated with that transaction. Each TID is unique to a given server and is known only by that server. See also *GTID*, *transaction*.

TIDL. Transactional Interface Definition Language. A high-level language provided by Encina to facilitate the development of transactional remote procedure call (TRPC) interfaces. TIDL simplifies the writing of code using TRPCs. It is an extension of the Distributed Computing Environment (DCE) Interface Definition Language (IDL). The TIDL compiler is named **tidl**. See also *IDL*, *stub*, *TRPC*.

timeout. A time limit on an action. If the action does not occur in the specified time period, it is abandoned.

T-ISAM. The Encina Transactional ISAM interface. This interface conforms to the X-ISAM interface and provides compatible functions for users of the Informix C-ISAM interface. T-ISAM also offers support for multithreaded applications and distributed transactions.

TM-XA. Transaction Manager-XA Service. A component of the Encina Toolkit Server Core that implements the transaction manager side of the X/Open XA interface for coordinating distributed transactions with XA-compliant resource managers. See also *resource manager*, *Toolkit Server Core*, *XA-compliant resource manager*.

Toolkit. A component of Encina that extends the services of the Distributed Computing Environment (DCE) with core technologies that enable transactional integrity and recoverability. The Encina Toolkit consists of the Executive and Server Core. All Encina applications use the Toolkit. Some applications use the Toolkit interfaces directly; others use them indirectly, via higher-level interfaces such as the Monitor or RQS. See also *Toolkit Executive*, *Toolkit server*, *Toolkit Server Core*, *Tran-C*.

Toolkit Executive. The set of Encina Toolkit components that, together, provide services that enable a process to initiate, participate in, and commit distributed transactions. The primary components of the Executive include the Transaction Service (TRAN), the Transactional Remote Procedure Call Service (TRPC), and the Thread-to-Tid Mapping Service (ThreadTid). See also *ThreadTid*, *Toolkit*, *Toolkit Server Core*, *TRAN*, *Tran-C*, *TRPC*.

Toolkit server. A server that uses the components of the Encina Toolkit. The term Toolkit server is used to identify servers that use the Encina Toolkit but not the Monitor application programming interface (API). Servers that use the Monitor API are referred to as Monitor application servers. Encina RQS, SFS, and PPC gateway servers are examples of Toolkit servers. See also *application server*, *recoverable server*, *Toolkit*.

Toolkit Server Core. The set of Encina Toolkit components that, together, provide facilities that enable a server process to manage recoverable data. The primary components of the Server Core include the Lock Service (LOCK), the Log Service (LOG), the Recovery Service (REC), the Transaction Manager-XA Service (TM-XA), and the Volume Service (VOL). See also *LOCK*, *LOG*, *REC*, *recovery service*, *TM-XA*, *Toolkit*, *Toolkit Executive*, *Toolkit server*, *VOL*.

top-level transaction. A transaction that does not execute within the scope of another transaction. A top-level transaction is the root of a transaction family, even if it is the only transaction in the family. See also *nested transaction*, *transaction*.

TPN. PPC: Transaction Program Name. A name that specifies either partner in a conversation. A TPN is generally used to refer to the program that accepts a conversation. It is analogous to a remote procedure in a remote procedure call (RPC) application. See also *conversation*.

trace mask. A filter for controlling the type of event tracing that is enabled for a component. For example, in the Recovery Service (REC) component, a trace mask can be used to trace only media recovery events.

TRAN. Distributed Transaction Service. A component of the Encina Toolkit Executive that coordinates multiple transactions, guarantees that transactions either commit or abort, and manages the delivery of information about transaction outcome to the participants of the transaction. See also *distributed transaction*, *Toolkit Executive*, *transaction*.

Tran-C. Transactional-C. A C-language programming interface designed to simplify the development of Encina transactional applications. Tran-C defines functions and constructs that provide high-level interfaces to functionality that is commonly used in transactional programming. See also *Toolkit*.

Tran-C++. Transactional-C++. A C++ programming language interface designed to simplify the development of Encina object-oriented transactional applications. See also *Tran-C*.

transaction. A set of operations that transforms data from one consistent state to another. The operations must be executed together and depend on each other for correctness. A few common examples of operations making up a transaction are requests for existing information, requests to modify existing information, requests to add new information, or any combination of the three. See also *abort*, *commit*, *distributed transaction*, *roll back*, *transaction processing system*.

Transactional Indexed Sequential Access Method. See *T-ISAM*

transaction coordinator. A participant in a transaction that manages prepare and commit responses from all other participants in the transaction. See also *coordinator migration*, *transaction*.

transaction factory. An object designed to create transaction objects. See also *factory*.

transaction family. All nested transactions that share a common ancestor. All members of a transaction family commit together and drop their locks simultaneously. See also *top-level transaction*, *transaction*.

transaction ID. See *TID*.

Transaction Manager-XA Service. See *TM-XA*.

transaction processing system. A system that maintains the properties of atomicity, consistency, isolation, and durability for distributed transactions. A distributed transaction processing system features recoverable processes and a commit protocol. See also *ACID properties*, *commit protocol*, *recoverable process*, *transaction*, *two-phase commit protocol*.

transaction program. PPC: A program that communicates via conversations. See also *conversation*.

Transaction Program Name. See *TPN*.

Transactional Interface Definition Language. See *TIDL*.

transactional remote procedure call. See *TRPC*.

Transactional Remote Procedure Call Service. See *TRPC*.

Transactional-C. See *Tran-C*.

transparent binding. Monitor: A form of binding that allows a Monitor client application to bind to any available server that exports a desired interface; the client does not need to explicitly specify a server. See also *explicit binding*.

TRPC. (1) Transactional remote procedure call (RPC). An RPC that carries additional information to identify the transaction on whose behalf it is executing. A TRPC is similar to a standard RPC. Like an RPC, a TRPC looks similar to a local procedure call to the calling program. See also *RPC*. (2) Transactional Remote Procedure Call Service (TRPC). A component of the Encina Toolkit Executive that provides the underlying communications mechanisms used by

the entire system. The service enables transactions to be distributed to other programs and nodes. See also *Toolkit Executive*.

two-phase commit protocol. A commit protocol for distributed transaction processing systems. The two-phase commit protocol has a prepare phase and a resolution phase. In the prepare phase, each participant in a transaction either guarantees to commit or aborts (reports that it cannot guarantee to commit). In the resolution phase, the transaction commits if all participants can commit or aborts if any of the participants cannot commit. See also *prepare phase*, *resolution phase*, *transaction processing system*.

type hierarchy. Monitor: The tree-structured collection of object types that classifies the objects in the Monitor repository. See also *ancestor*, *child*, *descendant*, *inheritance*, *parent*, *repository*, *root type*.

U

unique index. SFS: A file index with key values that differ from each other; that is, the index contains no duplicate key values.

universal unique identifier. See *UUID*.

unmanaged node. Monitor: A node (machine) that does not run a node manager. Client applications can run on unmanaged nodes. See also *managed node*.

unmarshaling. The process of unbundling the return packets from a remote procedure call (RPC) to extract the response from the remote process. See also *marshaling*, *RPC*.

upcall. A function registered with other functions in the Toolkit by an Encina application to request that an action take place whenever a specified event occurs. Unlike callbacks, upcalls are necessary to synchronize the events required by the transaction system; once registered, upcalls apply to every subsequent transaction. See also *callback*.

upgrade lock. A lock that announces the potential need to modify protected data. An application obtains an upgrade lock instead of a read lock to reduce the likelihood of a subsequent request for a write lock becoming deadlocked.

URL (Uniform Resource Locator). The address convention that indicates the location of an item on the World Wide Web. It includes the protocol followed by the fully qualified hostname and the request. The server typically maps the request portion of the URL to a path and filename. For example, **http://www.transarc.com/index.html**.

user exception. An exception defined and thrown by a user application. See also *system exception*.

UUID. Universal unique identifier. A hexadecimal number that uniquely identifies an entity such as a principal, a remote procedure call (RPC) interface, or an RPC object across all Distributed Computing Environment (DCE) cells. See also *DCE cell*.

V

variable-length field. RQS and SFS: A field of varying length in an RQS element or SFS record that contains data prefaced by an internal, opaque field specifying the length of the field.

VOL. Volume Service. A component of the Encina Toolkit Server Core that provides a logical interface to underlying physical storage. It enables volumes and files to span multiple physical devices. See also *Toolkit Server Core*, *volume*.

volume. An abstract representation of disk space used for storage by Encina components. Toolkit servers use Toolkit volumes, which include a logical volume and an underlying physical volume. A physical volume refers to physical regions on one or more disks; the regions need not be contiguous. A logical volume manages a physical volume by controlling all input/output and the mounting and unmounting of the physical volume. See also *logical volume*, *physical volume*.

Volume Service. See *VOL*.

W

warm start. A restart of a server. During a warm start, volumes are automatically recovered. See also *cold start*.

weighted prioritization. RQS: A queue set configuration in which dequeueing is distributed among priority classes based on service levels. A nonzero service level defines a weight for each class. The relationship between the weights determines the ratio of dequeueing chances given to the priority classes. See also *queue*, *starvation*, *strict prioritization*.

well-known endpoint. A preassigned, stable address that a server process uses every time it runs. Well-known endpoints are exported to the namespace. See also *dynamic endpoint*, *endpoint*.

work. RQS: The volume of business associated with a queue. The interpretation of the work (for example, dollars transferred or tons produced) varies with the queue. As an element is enqueued, its work value is added to an accumulating work sum. See also *accumulation*, *queue*.

write lock. An exclusive lock that prevents concurrent access to the locked data. See also *exclusive lock*.

write-ahead logging. A logging mechanism that causes all log records associated with a transaction to be written to the log before the transaction actually commits. This guarantees that the records are present in the log and can be used to restore recoverable data to a correct state if the system fails at the exact time the transaction commits. See also *logging*.

X

XA-compliant resource manager. A resource manager that uses the X/Open Distributed Transaction Processing XA interface. XA-compliant resource managers allow Encina to coordinate distributed transactions. See also *native resource manager*, *resource manager*, *TM-XA*.

Notices

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

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

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

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

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will

be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

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

IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222
U.S.A.

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

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

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

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

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

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

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and service marks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States, other countries, or both:

Advanced Peer-to-Peer Networking
AFS
AIX
APPN
AS/400
CICS
CICS OS/2
CICS/400
CICS/6000
CICS/ESA
CICS/MVS
CICS/VSE
CICSPlex
DB2
DCE Encina Lightweight Client
DFS
Encina
IBM
IBM System Application Architecture
IMS
IMS/ESA
Informix
Language Environment
MQSeries
MVS
MVS/ESA
NetView
Open Class
OS/2
OS/390
OS/400
Parallel Sysplex
PowerPC
RACF
RAMAO
RMF
RISC System/6000
RS/6000
S/390
SAA
SecureWay
TeamConnection
Tivoli
TXSeries
VSE/ESA
VTAM
VisualAge
WebSphere

Domino, Lotus, and LotusScript are trademarks or registered trademarks of Lotus Development Corporation in the United States, other countries, or both.

ActiveX, Microsoft, Visual Basic, Visual C++, Visual J++, Visual Studio, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Pentium is a trademark of Intel Corporation in the United States, other countries, or both.

This software contains RSA encryption code.

Other company, product, and service names may be trademarks or service marks of others.

Index

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W