z/OS

Encina Transactional RPC Support for IMS™

SC24-5920-00

z/OS

Encina Transactional RPC Support for IMS™

SC24-5920-00

Note

Before using this information and the product it supports, be sure to read the general information under Appendix B, “Notices” on page 39.

First Edition (March 2001)

This edition, SC24-5920-00, applies to Version 1 Release 1 of z/OS Encina (program number 5694-A01), and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. A form for reader's comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Information Development, Dept. G60
1701 North Street
Endicott, NY 13760-5553
United States of America

FAX (United States & Canada): 1+607+752-2327
FAX (Other Countries):
Your International Access Code +1+607+752-2327

IBMLink™ (United States customers only): GDLVME(PUBRCF)
Internet e-mail: pubrcf@vnet.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

About This Book

This book is intended to help software designers and programmers extend their IMS transaction applications to participate in a distributed, transactional client/server application. This client/server application includes an IMS application, which is invoked by DCE Application Support, that uses Encina and DCE to handle the complexities of large distributed systems and to maintain data integrity across them. This book describes the resources and facilities needed to access Encina resources from an IMS application and provides step-by-step instructions describing how to develop an IMS transaction application using Encina transactional support.

Who Should Use This Book

This book is intended for system designers; IMS application designers, programmers, and administrators; and DCE designers and programmers who develop and administer client/server applications.

What You Need to Know

You need to understand:

- The basic concepts of DCE
- The z/OS operating environment
- Transaction processing concepts
- The concepts and practice of the transactional remote procedure call
- IMS transaction programming.

Also, system and network administrators need to understand how to use the administration facilities DCE provides. Designers and programmers need to be familiar with the DCE Interface Definition Language (IDL) and C.

What This Book Contains

The following section briefly describes the format and organization of this book.

Chapter 1, “What is IMS Transactional RPC Support?” on page 1 introduces the concepts of IMS transactional RPC support and lists the software dependencies you need to use the IMS transactional RPC support.

Chapter 2, “Installation” on page 5 lists the IMS transactional RPC support parts shipped in HFS and describes the installation verification program (IVP).

Chapter 3, “Writing an IMS Transactional RPC Application” on page 11 describes how to write an application that issues IMS transactional RPC calls to a remote Encina server and ensures data integrity. This chapter also describes how to create a Transactional Interface Definition Language (TIDL) file to define your interfaces.

Chapter 4, “Building an IMS Transactional RPC Application” on page 23 provides the steps to build an IMS transactional RPC application.

Chapter 5, “Managing Transactions” on page 31 describes the tasks that are involved in managing transactions called by IMS transactional RPC applications.

Appendix A, “IMS Transactional RPC Programming Interfaces” on page 33 describes the format of the application programming interfaces your IMS application must use to perform transactional RPCs using Encina support.

Conventions Used in This Book

This book uses the following typographic conventions:

Bold	**Bold** words or characters represent system elements that you must enter into the system literally, such as commands, options, or path names.
Italic	*Italic* words or characters represent values for variables.
`Example font`	Examples and information displayed by the system appear in `constant width type style`.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices.
< >	Angle brackets enclose the name of a key on the keyboard.
...	Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.
\	A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character **** as the last non-blank character on the line to be continued, and continue the command on the next line.

This book uses the following keying conventions:

<**Alt-***c*>	The notation <**Alt-***c*> followed by the name of a key indicates a control character sequence.
<**Return**>	The notation <**Return**> refers to the key on your keyboard that is labeled with the word Return or Enter, or with a left arrow.
Entering commands	When instructed to enter a command, type the command name and then press <**Return**>.

Where to Find More Information

Where necessary, this book references information in other books using shortened versions of the book title. For complete titles and order numbers of the books for all products that are part of z/OS, see the *z/OS Information Roadmap*, SA22-7500. For complete titles and order numbers of the books for z/OS DCE, refer to the publications listed in the “Bibliography” on page 53.

For information about installing z/OS Encina components, see the *z/OS Program Directory*.

For information about DCE application programming tasks, see:

- *z/OS DCE Application Development Guide: Introduction and Style*, SC24-5907
- *z/OS DCE Application Development Guide: Core Components*, SC24-5905
- *z/OS DCE Application Development Guide: Directory Services*, SC24-5906

For information using z/OS UNIX System Services commands to perform end-user tasks, see:

- *z/OS UNIX System Services User's Guide*, SA22-7801.

For information about the Application Support (AS) server, see:

- *z/OS DCE Application Support Configuration and Administration Guide*, SC24-5903
- *z/OS DCE Application Support Programming Guide*, SC24-5902

For information about using IMS/ESA® see:

- *IMS/ESA Administration Guide: System*, SC26-8730
- *IMS/ESA Administration Guide: Transaction Manager*, SC26-8731
- *IMS/ESA Customization Guide*, SC26-8732
- *IMS/ESA Operations Guide*, SC26-8741
- *IMS/ESA Operator's Reference*, SC26-8742
- *IMS/ESA Application Programming: Design Guide*, SC26-8728
- *IMS/ESA Open Transaction Manager Access Guide*, SC26-8743

For information about configuring DCE for use with the Encina Toolkit and administering an Encina Toolkit server, see:

- *z/OS Encina Toolkit Executive Guide and Reference*, SC24-5919

For help in handling errors, see:

- *z/OS Language Environment Debugging Guide*, GA22-7560

Softcopy Publications

The z/OS Encina library is available on a CD-ROM, *z/OS Collection*, SK3T-4269. The CD-ROM online library collection is a set of unlicensed books for z/OS and related products that includes the IBM Library Reader.™ This is a program that enables you to view the BookManager® files. This CD-ROM also contains the Portable Document Format (PDF) files. You can view or print these files with the Adobe Acrobat reader.

Internet Sources

The Softcopy z/OS publications are also available for web-browsing and for viewing or printing PDFs using the following URL:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/`

You can also provide comments about this book and any other z/OS documentation by visiting that URL. Your feedback is important in helping to provide the most accurate and high-quality information.

Using LookAt to Look up Message Explanations

LookAt is an online facility that allows you to look up explanations for z/OS messages. You can also use LookAt to look up explanations of system abends.

Using LookAt to find information is faster than a conventional search because LookAt goes directly to the explanation.

LookAt can be accessed from the Internet or from a TSO command line.

You can use LookAt on the Internet at:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookat.html`

To use LookAt as a TSO command, LookAt must be installed on your host system. You can obtain the LookAt code for TSO from the LookAt Web site by clicking on the **News and Help** link or from the *z/OS Collection*, SK3T-4269.

To find a message explanation from a TSO command line, simply enter: **lookat** *message-id* as in the following:

```
lookat iec192i
```

This results in direct access to the message explanation for message IEC192I.

To find a message explanation from the LookAt Web site, simply enter the message ID and select the release with which you are working.

Note: Some messages have information in more than one book. For example, IEC192I has routing and descriptor codes listed in *z/OS MVS Routing and Descriptor Codes*, SA22-7624. For such messages, LookAt prompts you to choose which book to open.

Accessing Licensed Books on the Web

z/OS licensed documentation in PDF format is available on the Internet at the IBM Resource Link site:

`http://www.ibm.com/servers/resourcelink`

Licensed books are available only to customers with a z/OS license. Access to these books requires an IBM Resource Link user ID, password, and z/OS licensed book key code. The z/OS order that you received provides a memo that includes your key code.

To obtain your IBM Resource Link user ID and password, logon to:

`http://www.ibm.com/servers/resourcelink`

To register for access to the z/OS licensed books:

1. Logon to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.
3. Select **Access Profile**.
4. Select **Request Access to Licensed books**.
5. Supply your key code where requested and select the **Submit** button.

If you supplied the correct key code you will receive confirmation that your request is being processed.

After your request is processed you will receive an e-mail confirmation.

Note: You cannot access the z/OS licensed books unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

To access the licensed books:

1. Logon to Resource Link using your Resource Link user ID and password.
2. Select **Library**.
3. Select **zSeries**.
4. Select **Software**.
5. Select **z/OS**.
6. Access the licensed book by selecting the appropriate element.

Chapter 1. What is IMS Transactional RPC Support?

An IMS transactional RPC application is part of an Encina distributed, transactional client/server application. It uses Encina and DCE to handle the complexities of large distributed systems and to maintain data integrity across them.

This chapter introduces you to concepts of the IMS transactional RPC support, describes the terminology this book uses and the software requirements needed to run your IMS transactional RPC application.

IMS transactional RPC support enables an IMS transaction to:

- Update IMS data, and
- Issue one or more Encina transactional RPC calls in the same unit of work to update data on remote Encina servers

while ensuring that the updates to both Encina and IMS resources are consistent (all updates are either committed or aborted).

With this support, the business logic to access Encina resources can reside in the IMS transaction application or an Encina client application. This gives you the flexibility to access data within a distributed environment.

Concepts and Terminology

Figure 1 illustrates the components required to set up your Encina transactional, distributed, client/server application. Following the figure is a description of each of the components and a list of terms used throughout this book.

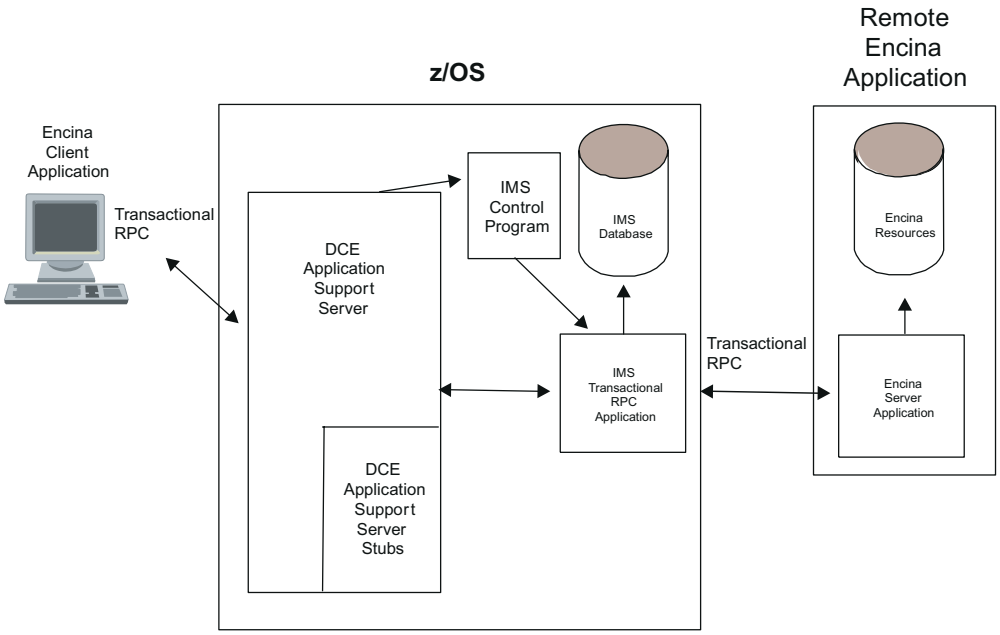

Figure 1. Components of a Transactional, Distributed, Client/Server Application using Encina Transaction Support for IMS

DCE Application Support Server
A z/OS DCE Application Support (AS) server that transforms requests from DCE clients into a form CICS® or IMS regions can understand. Within this book, AS refers to AS IMS with OTMA enabled for transactional RPCs.

DCE Application Support Server Stub
An executable that is generated by the z/OS IDL compiler using the DCE extended IDL language. The application support server stub accepts input from the client, generates input that IMS will accept, and formats return information for the client.

Encina
A family of products layered on top of DCE that enhance DCE RPCs with transactional semantics. Encina implements a two-phase commit protocol for transactional RPCs. This is a set of actions that ensures an application program makes *all* the changes of a transaction to a collection of resources or makes *none* of the changes.

Encina client application
The application that initiates the IMS transactional RPC application through AS IMS.

Encina server application
An application that accepts requests from another Encina application; it must include a TRAN service. If the server program manages permanent storage and is expected to be restarted after termination or failure, it must have a recovery service.

IMS Transactional RPC Application
An IMS transaction application that issues Encina transactional RPCs. From Encina's point of view, it is an Encina client because it uses the client stubs. This application cannot use any Encina recoverable resources.

IMS (Information Management System)
Information Management System/Enterprise Systems Architecture Database Manager (IMS DB) and Transaction Manager (IMS TM) are licensed programs that run under the Multiple Virtual Storage/Enterprise Systems Architecture (MVS/ESA™) operating system. IMS DB is a database system. IMS TM is a data communication system. Together they create a complete online transaction processing environment providing continuous availability and data integrity.

z/OS Encina Toolkit Executive
A subset of Transarc's Encina. Although the z/OS Encina Toolkit Executive supports only ephemeral clients (clients that provide no facility for logging results or recovery), Application Support provides recovery by using native z/OS Resource Recovery Services (RRS) and IMS facilities in combination with Toolkit services.

transaction
Refers to an Encina transaction (unless specifically called an IMS transaction). An Encina *transaction* is a set of operations that must be done as a single unit for consistent transformation of data. 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**. Once a transaction commits, its effects are permanent. A subsequent failure does not alter the transaction's effects.

transactional RPC
Similar to a standard DCE RPC, but a transactional RPC includes additional information that identifies the Encina transaction on whose behalf it is running. A transactional RPC uses DCE RPCs as its underlying communication mechanism, but extends this by providing transactional semantics.

Software Dependencies

Minimum z/OS Software Dependencies

- IMS/ESA 6.1

  Enabled for OTMA.

- z/OS
  - z/OS Resource Recovery Services
  - z/OS DCE
  - z/OS DCE Application Support for IMS

    Configured for OTMA and transactional RPCs.
  - z/OS Encina Toolkit Executive
  - z/OS UNIX System Services.

Encina Client and Server Dependencies

- OSF Distributed Computing Environment
- *Transarc* Encina Toolkit Executive (client and server)
- *Transarc* Encina Server Core (server).

Chapter 2. Installation

The IMS transactional RPC support is part of z/OS. If you choose to install the z/OS Server Pack, you will not need to install the IMS transactional RPC support separately. If you choose the z/OS product delivery offering (PDO), you can install the IMS transactional RPC support using SMP/E.

The Program Directory contains the directions for installing the IMS transactional RPC support using SMP/E. The following information details where the installed files reside.

z/OS Encina Toolkit Executive and IMS Transactional RPC Support Parts Shipped in HFS

- /usr/lpp/encina/bin

  Contains executable files, for example, the TIDL compiler
- /usr/lpp/encina/etc

  Contains executable files and scripts, for example, trace formatting routines
- /usr/lpp/encina/include

  Contains header files
- /usr/lpp/encina/lib

  Contains libEncina.x
- /usr/lpp/encina/msg/C

  Contains posix format symbol and trace preprocessor files
- /usr/lib/msg

  Contains the message catalogs
- /usr/lpp/encina/example/merchandise

  Contains the Merchandise example program and the Encina Merchandise Installation Verification Procedure

Understanding the IMS Transactional RPC Support Installation Verification Program

The Encina Merchandise Installation Verification Program (IVP) verifies that IMS transactional RPC support is installed correctly. Before starting the IVP for IMS transactional RPC support, we recommend that you complete the IVP for the z/OS Encina Toolkit Executive and AS IMS. See the *z/OS Encina Toolkit Executive Guide and Reference* for IVP information on the z/OS Encina Toolkit Executive. See the *z/OS DCE Application Support Configuration and Administration Guide* for IVP information on AS IMS.

In the IVP, all parts are designed to be built and run on z/OS. The Encina client application and the Encina server application are run using z/OS UNIX System Services. Sample files are shipped with the product. You will use these files to set up and run the IVP. These sample programs provide you with a simple order/query, client/server application. Figure 2 on page 6 illustrates the flow between the different components involved in this merchandise application.

After you verify that IMS transactional RPC support is installed correctly, you will be ready to write your IMS transactional RPC source program and build your IMS transactional RPC source program.

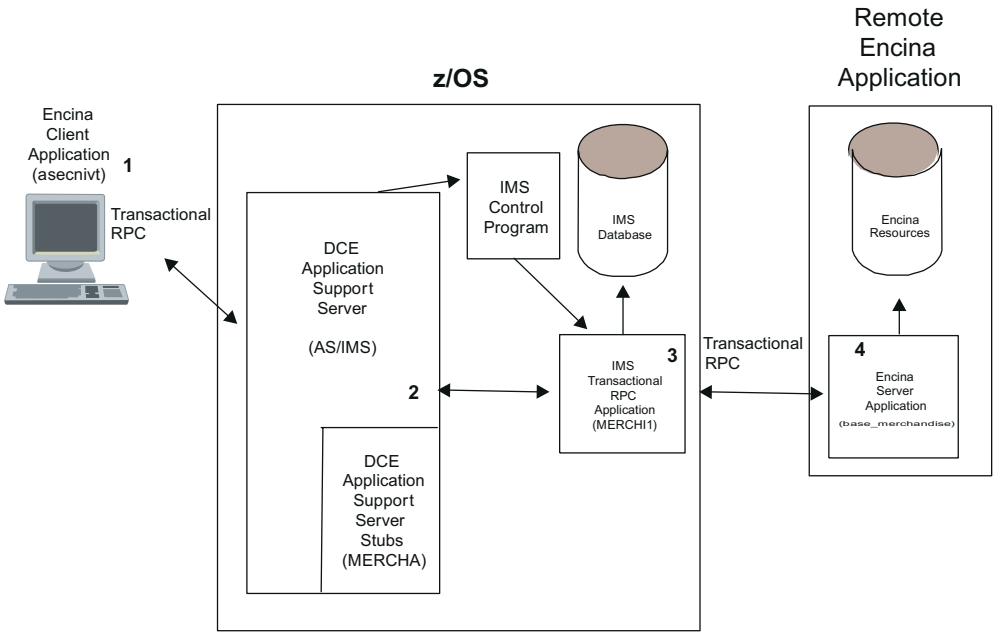

Figure 2. Components of the IVP for IMS Transactional RPC Support

The following are components that are part of the processing of the IVP for IMS transactional RPC support.

1 A DCE client application (**asecnivt**) is issued from z/OS UNIX System Services. This application issues requests using transactional RPCs to Application Support/IMS. The IVP source file (**asecnivt.c**) and all the files required to build an Encina client application (**asecnivt**) are shipped with the product and are located in `/usr/lpp/encina/example/ecnims/asimso`.

asecnivt represents a client application that can be located anywhere in the DCE environment.

2 AS IMS transforms the request from the **asecnivt** Encina application into an IMS request and inserts the request into the IMS message queue. The rules for this transformation are based on the definitions in the **mercha.idl** file. The DCE IDL compiler compiles the **mercha.idl** file to create the **MERCHA** DCE AS server stubs.

3 The IMS application (**MERCHI1**) issues a transactional RPC to access an Encina server application (**base_merchandise**). The interface between the IMS transactional RPC application and the Encina server application is defined in **merchandise.tidl**.

The source file (**merchi1.c**) and all the files required to build an IMS transactional RPC application (**MERCH1I**) are shipped with the product and are located in `/usr/lpp/encina/example/ecnims/imstran`.

Most IMS transactional RPC applications can update a database that is controlled by IMS. Our IVP application does not perform the database updates.

4 The **base_merchandise** server application updates Encina resources. This sample program is also part of the z/OS Encina Toolkit Executive IVP.

base_merchandise represents a server application that can be located anywhere in the DCE environment.

Setting Up the IVP

IMS Administrator

1. Contact your IMS administrator to define the **MERCHI1** program to IMS. See the README file in the `/usr/lpp/encina/example/ecnims/imstran` directory for transaction and program definitions.
2. The MVS user ID corresponding to the started task for the IMS processing region must have a UNIX ID (uid) and home directory associated with it. Using the /SECURE command, you must set OTMA security to a value other than FULL. See the *IMS/ESA Operator's Reference* for details on the /SECURE command.

Application Programmer

1. Building the **MERCHI1** (IMS transactional RPC application) and **base_merchandise** (Encina server application).
   a. Create an `IVT/imstran` directory.
   b. Copy all the files from the `/usr/lpp/encina/example/imstran` directory into that directory.
   c. Edit the Makefile and set the following names:

      YOUR_IMS_PGMLIB=*pgmlibname*
      pgmlibname is the name of the PDS where the IMS transaction program is to be built. (This dataset name should be obtained from your IMS administrator).

      IMSRESLIB="//'*name*'
      name is the dataset name where your IMS RESLIB dataset is placed.

   d. From the `IVT/imstran` directory, enter the following command:

      ```
      make -f Makefile
      ```

      The Makefile builds **MERCHI1** as an executable member of the PDS file (*pgmlibname*) and **base_merchandise** as an executable HFS file in the IVT directory.

      Note: base_merchandise may already be running because it was built in the IVP for the z/OS Encina Toolkit Executive. You can rebuild it again here.

2. Building **asecnivt** (Encina client application) and **MERCHA** (AS server stubs).
   a. Create an `IVT/asimso` directory.
   b. Copy all the files from the `/usr/lpp/encina/example/asimso/` directory into that directory.
   c. Edit the Makefile and set the variable PDS_NAME to the name of the partitioned data set where the AS IMS interface stub will reside. Also, edit the Makefile.client as needed.
   d. From the `IVT/asimso` directory, enter the following command:

      ```
      make -f Makefile
      ```

      The Makefile builds the Application Support interface stub as a member of the partitioned dataset you have specified in the Makefile. It also builds the Encina client application, **asecnivt**, that communicates with AS IMS.

AS IMS Administrator

It is assumed that the AS IMS server has been configured as described in the *z/OS DCE Application Support Configuration and Administration Guide*. It is also assumed that the AS IMS Administrator is a DCE principal ID that has been given access to the administration functions of the AS IMS server, and the DCE principal ID "telshop-prin" has been created (as part of the z/OS Encina Toolkit Executive IVP). For

details of defining DCE principal identities and setting their authorizations for the Application Support servers, see the *z/OS DCE Application Support Configuration and Administration Guide*.

1. Start the Application Support server by submitting your AS IMS server job-name.
2. From the TSO command line within ISPF, type 'ASUADMIN' to run the AS administration program.
3. Enter the CDS name of the AS IMS server you have started at the top of the panel.
4. Login to DCE as the principal name authorized to perform administrative actions on the server. You may do this from option 6 of the ASUADMIN panel.
5. Step 6 cannot begin until the AS IMS server is in quiescent state. Select option 3 on the second panel to view the status of the AS IMS server. Exiting the screen and selecting option 3 again on the second panel updates the screen. When the server state is quiescent, return to the main panel.
6. Select option 1 from the main menu to perform a start attachment operation.
7. Make sure the **MERCHA** interface is installed by selecting option 3 from the ASUADMIN panel to view server operational data. From that panel, select option 1 to display installed interfaces.

   If **MERCHA** is not present, return to the main panel and select option 4 to install interfaces. From the install/uninstall panel, enter an action code of 1 for interface name **MERCHA**, and press Enter.
8. Update your mapping of DCE Principal names to MVS user IDs to map the "telshop-prin" DCE principal name to an MVS User ID. Depending on the way your Application Support server is defined, you must use either the Application Support Identity Mapping file or RACF®, a component of the SecureWay® Security Server for z/OS. See the *z/OS DCE Application Support Configuration and Administration Guide* for details.

Running the IVP

1. dce_login as merch-prin.
2. Change to the `IVT/imstran` directory that contains the **base_merchandise** executable application.
3. Start the **base_merchandise** server in the background:

   ```
   base_merchandise /.:/encina/examples/merch-entry   merch-prin
                       /tmp/merch-prin.ktf   null   null   &
   ```

   The parameters are:

   `/.:/encina/examples/merch-entry`
   is the CDS entry name

   `merch-prin`
   is the principal name

   `/tmp/merch-prin.ktf`
   is the keytab file name

   `null  null  &`
   used for parameters not needed in z/OS
4. Change to the `IVT/asimso` directory that contains the **asecnivt** executable application.
5. Start **asecnivt**.

   `asecnivt` *`AS_IMS_server_name`*

AS_IMS_server_name
is the CDS name of the AS IMS server. For example, /.:/AS/server1.

6. When prompted for commands, type ? for a list of valid commands. The following is the output from a sample run:

```
asecnivt <AS_IMS_server_name>
   Begin_or_End:  ?

 Valid commands are:
    b: Begin an order.
    e: Exit the program.
    ?: Prints a help message.

    Begin_or_End: b

      Command: ?

    Valid commands are:
     i:  Query an item's availability through AS
     j:  Order a quantity of some item through AS
     c:  Commit an order
     a:  Abort. End this order without committing it.
     ?:  Prints a help message.

       Command: i

     Check item: 54
     There are 10 of item 54

       Command: c
   Order processed.
     Begin_or_End: e
```

7. The base_merchandise server runs indefinitely in the background. You should stop it to free resources and avoid problems. To stop the server:
   a. Determine the *pid*:

      ```
      ps -lf
      ```

   b. Enter:

      ```
      kill -9 <pid>
      ```

8. Clean up after the sample program has been called.
   a. Uninstall the MERCHA interface by using ASUADMIN option 4, specifying an action code of 0 for MERCHA, and pressing Enter.
   b. Return to the main panel of ASUADMIN, and select option 2 to stop the attachment to the server.
   c. Exit the ASUADMIN program.
   d. Stop the AS IMS server by issuing the MVS console command:

      ```
      /STOP <job-name>
      ```

Chapter 3. Writing an IMS Transactional RPC Application

An IMS transactional, distributed client/server RPC application consists of at least two interface programs and four other components—Encina Client Application, Application Support Server, IMS Transactional RPC Application, and Encina Server Application (see Figure 1 on page 1).

The IDL file defines the interface between the Encina client application and the DCE Application Support Server. z/OS DCE IDL has extensions to support Encina semantics for AS. Refer to the *z/OS DCE Application Support Programming Guide* for information on writing the IDL file, the Encina client application, and the Application Support Server application.

The TIDL file defines the transaction interface between the IMS transactional RPC application and the remote Encina server. This chapter describes how to write a TIDL file and the IMS transactional RPC application. Refer to the Encina documentation for the platform on which you are developing for information on writing an Encina server application. Refer to *z/OS Encina Toolkit Executive Guide and Reference* for information on writing the Encina server application (**base_merchandise**) shipped with the product.

High-Level Phases of an IMS Transactional RPC Application

Figure 3 on page 12 illustrates the high-level phases in developing an Encina application. The following steps explain how these phases are processed for an IMS transactional RPC application. Throughout these steps, “client” refers to the IMS transactional RPC application.

1. For the client and server: Write the TIDL code to define the interface.
2. For the server: Provide the code to set up and listen for requests.
3. For the client: Establish the Encina transaction environment using the **etran_Env_Init** API.

   In an IMS transactional RPC application, use **etran_Env_Init** to establish the Encina transaction environment.
4. For the client: If you wish to do an authenticated DCE (or Encina) request, your client application must get DCE credentials.

   In an IMS transactional RPC application, issue a DCE login and related APIs to obtain DCE credentials. See the *z/OS DCE Application Development Guide: Core Components* discussion under the **login** application interface.

   If the transaction is one that waits for input (WFI) **and** you plan to use the same DCE login context across multiple calls of the transaction, keep in mind:

   - Your DCE credentials are accessible only from the MVS user ID that created them. If your installation is using the IMS Build Security Environment Exit (DFSBSEX0) to build a new security environment or your OTMA security setting is set to FULL, you must get a new set of credentials for each invocation of the transaction.
   - DCE credentials can expire during the lifetime of your transaction. See the *z/OS DCE Application Development Guide: Core Components* for information about refreshing your credentials.
5. For the client: Obtain a binding to the server.

   Create a binding handle for each remote server you plan to access.

   If you are doing an authenticated RPC, call **rpc_mgmt_inq_server_princ_name** and **rpc_binding_set_auth_info** to set the authentication information in the binding handle.

Call **trpc_ConsBinding** to create a trpc binding handle from the rpc handle.

6 For the client: Activate the Encina transaction.

In an IMS transactional RPC application, use **etran_Begin** to activate the Encina transaction. You must issue this before calling any transactional RPC.

7 For the client: Issue the transactional RPC call or calls. Handle the results and continue other work.

In an IMS transactional RPC application, if the transactional RPC caused an abort, no other work done by your IMS transaction will be made permanent.

However, you can explicitly issue a DL/1 or CPI-C request to abort all Encina transactions as well as IMS-controlled databases. Once you issue this rollback request, you cannot issue another transactional RPC for this Encina transaction.

8 For the server: Provide the code to service the requests.

9 For the client: Complete the Encina transaction and initiate a commit or abort operation.

The IMS transaction will complete either by ending the program or issuing a GU to the IMS message queue. IMS resolution is done outside the IMS transactional RPC application. The Encina client initiates a commit or abort that is first processed by AS IMS. Then z/OS RRS, IMS, and AS IMS all participate in the outcome resolution.

10 For the server: Process the transactional resolution (commit or abort)

11 For the client: Cleanup the transactional environment.

In an IMS transactional RPC application, use **etran_Term** to cleanup the transactional environment.

If you develop an IMS transactional RPC application, you must code all phases of the client except for 9 The IMS transaction resolution provides the Encina transaction outcome.

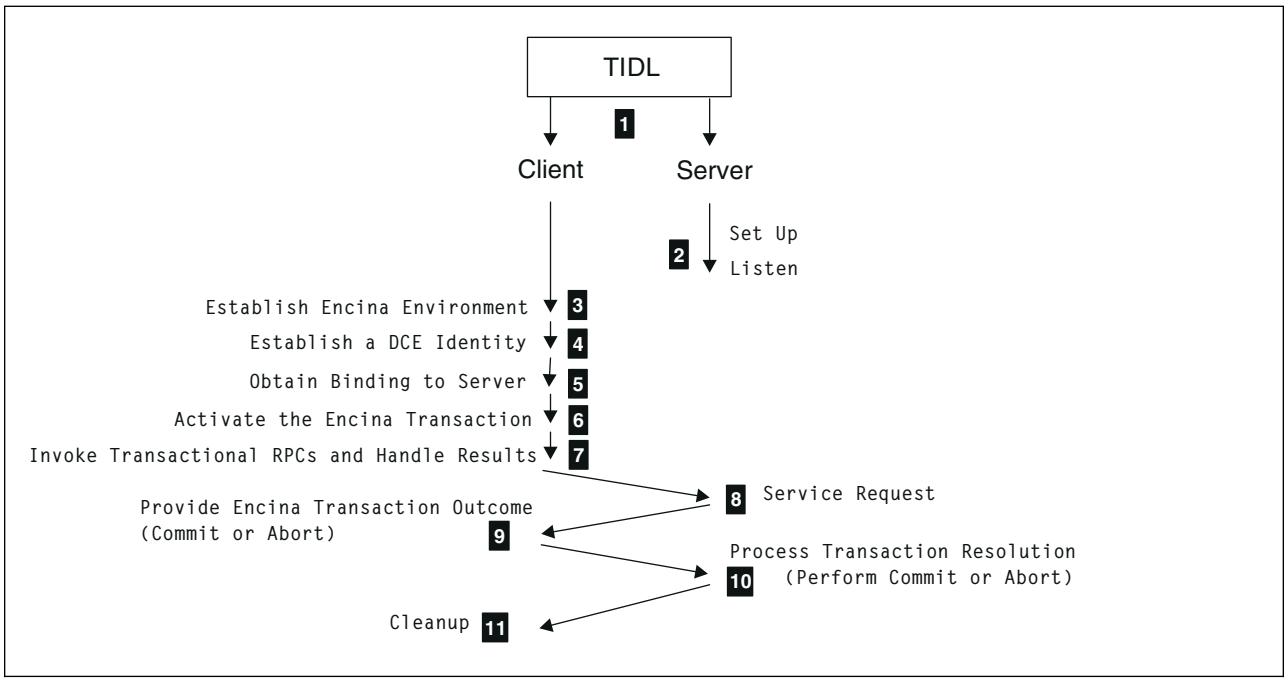

Figure 3. Phases in Developing an IMS Transactional RPC Application

Defining the Transactional Interface

One of the first steps in writing an IMS transactional RPC application is designing and writing the interfaces. This section provides a general overview of how to use TIDL to define interfaces for IMS transactional RPC applications that use the transactional RPC mechanism Encina provides. You must define the interfaces for a transactional application in a TIDL interface definition file before writing the client and server application code. After you define the interfaces, you can compile the TIDL interface definition file, using the TIDL compiler. The TIDL compiler is run with the **tidl** command. The TIDL compiler is a preprocessor that generates files containing stub code for the client and server applications.

For a complete description of transactional RPC, see the *z/OS Encina Toolkit Executive Guide and Reference*.

A TIDL compiler (**tidl**) uses two files to generate the stub and header files for use with DCE RPC:

*name***.tidl** A *transactional interface definition* (TIDL) file that contains the interface description for Encina's TIDL compiler to process. In addition to stub and header files, the compiler generates an IDL file to use as input to the DCE IDL compiler.

*name***.tacf** A *transactional attribute configuration* (TACF) file that specifies which of the operations defined in the associated TIDL file to export. Using an attribute configuration file lets the same TIDL file contain multiple definitions for an operation. You can selectively export operations for different applications by changing the TACF file. This file is optional.

Notes:

1. These files must be in the C programming language.
2. These files must reside in the Hierarchical File System (HFS) because the Encina Toolkit Executive TIDL preprocessor does not support MVS data sets (PDS or PDSE files).

The TIDL File: The TIDL file defines the interface for your transactional application. Figure 4 is the sample TIDL file, **merchandise.tidl**, that is shipped with the product. It is located in **/usr/lpp/encina/example/ecnims/imstran** directory. **merchandise.tidl** defines the **merchandise** interface that includes two functions, **merchandise_QueryItem** and **merchandise_OrderItem**.

```
[
uuid(0002068a4-f049-bdfc-c037cf6a0000),
version(1.0)
]

interface merchandise
{
[transactional] void merchandise_QueryItem(
                                [in] long stockNum,
                                [out] long *amountP);
[transactional] void merchandise_OrderItem(
                                [in] long stockNum,
                                [in] long amount);
}
```

Figure 4. merchandise.tidl

The first line contains a universal unique identifier (UUID) for this interface. You can use the DCE RPC **uuidgen** utility to generate a UUID as well as an empty TIDL or IDL template file that can serve as the basis for your specialized TIDL file.

Following the UUID is the version number of the interface. The version number is used in some of the internal data structure names TIDL and IDL produce, uniquely identifying the client and server stubs produced from a certain version of the TIDL input file. The form of the version number is

major-version-number.minor-version-number. If the version number is a single integer, TIDL interprets this as the major number and automatically supplies a minor number of 0.

Note: Transactional RPC does not support distinct, coexisting versions of the same interface.

The name of the interface (in this case **merchandise**) and the actual interface definition (that is, the functions that actually make up the interface), follow the UUID and version number information. The TIDL file must describe only functions requiring that information be exchanged using the RPC mechanism.

The **transactional** attribute prefixes the functions the TIDL file describes that are run within the scope of a transaction. In the sample **merchandise** interface, two parameters are defined for each of the functions in the TIDL file.

The TACF File: A TACF file can control the way binding occurs and the way errors and exceptions are reported. TACF files name the target interface and the operations in the interface that use modified binding and error handling. Figure 5 shows the TACF file associated with the **merchandise** TIDL file. The Merchandise example uses explicit binding, so the TACF file need not specify the type of binding. The TACF file specifies no operations, which means that neither binding nor error handling is changed for any of the operations defined in the corresponding TIDL file.

```
interface merchandise
{
}
```

Figure 5. merchandise.tacf

The TACF files that **tidl** uses have the same syntax as attribute configuration (ACF) files used with IDL files for DCE RPC. For more information about the syntax of ACF files, see *z/OS DCE Application Development Guide: Core Components* and *z/OS DCE Application Development Reference*.

Writing an IMS Transactional RPC Source Program

After you define the interface using TIDL, you are ready to write your IMS transactional RPC source program. You must code this program in the C language and use the IMS Transaction Manager.

Design Considerations for an IMS Transactional RPC Source Program

Keep in mind the following considerations when writing your IMS transactional RPC source program:

- An IMS transactional RPC source program is an IMS transaction program. An IMS message region processing an IMS transaction called as a result of a transactional RPC stays active until the global outcome is resolved. Any locks this IMS transaction holds are retained during this time.
- The overhead of initializing an IMS transactional RPC application is significant. We recommend you design your source program so that it will be processed multiple times when scheduled by IMS, therefore, initialization only occurs once. This recommendation will improve the total processing time of the IMS transactional RPC source program.
- If the remote server you are accessing requires authenticated RPCs, you must do a DCE login in the IMS transactional RPC source program. If your source program will be processed multiple times per schedule, it must run under the identity of the IMS region. This means your OTMA security setting must not be set to FULL. Refer to the *IMS/ESA Operator's Reference* for more information on OTMA security.
- All IMS transactional RPC API and DCE API calls must be written in the C language; however, they can be called from a subroutine of an IMS transaction program written in C, PL/I, or COBOL.

- You cannot issue transactional RPCs and APPC protected conversations from the same IMS transactional RPC source program.
- An IMS transactional RPC source program is a DCE program, which means they are compiled with posix(on) and they support most DCE APIs.

  An IMS transactional RPC source program is single threaded because IMS applications can only be single threaded.
- The Encina client application initiates Encina syncpoint processing. The Encina client initiates a commit or abort that is first processed by AS IMS. Then z/OS RRS, IMS, and AS IMS all participate in the outcome resolution.
- From an IMS transactional RPC source program, you can only use a small subset of the z/OS Encina Toolkit Executive. Specifically, your IMS transactional RPC source program can use the **trpc_ConsBinding** interface and the **trdce** operation. The TRDCE Utilities Library provides utilities for constructing DCE client and server programs. See the *z/OS Encina Toolkit Executive Guide and Reference* for more information on the TRDCE Utilities Library.

Sample Program Statements

You may want to use the sample program (**merchi1.c**) that is shipped with the product as a model. The sample program is located in the `/usr/lpp/encina/example/ecnims/imstran` directory. The following is a description of the statements used in the sample program (**merchi1.c**).

- **Set up the Language Environment® to handle IMS POSIX programs.**

```
#pragma runopts(env(IMS),plist(OS),posix(ON),stack(12000))
#pragma pack(packed)
```

- **Include the header files.**

  You include the header file the IDL compiler generated for your interface. The name of the header file depends on the name of your interface. Also include header files for all DCE and Encina facilities your program requires.

```
/* Include the following standard header files:  */

/*    For IMS C programs   */
#include <ims.h>

/*    IMS programs that use IMS transactional RPC APIs must include the        */
/*    ecnims.h file supplied by IMS transactional RPC support.   */
#include <ecnims/ecnims.h>

/     Supplied as z/OS Encina Toolkit Executive */
#include <trdce/trdce.h>

/*    For C language */
#include <ctype.h>
#include <string.h>

/*    Supplied by OS/390 DCE */
#include <dce/exc_handling.h>
#include <dce/rpcexc.h>

/* Include the header file the IDL compiler generated for your */
/* interface.  The name of the header file depends on the      */
/* name of your interface.  Change "merchandise" to the name   */
```

```
/* of your interface.                                          */
#include "merchandise.h"
```

- **Define DCE principal and password for DCE login.**

  You may want to use a more secure method to define a principal and password.

```
char *principal="telshop-prin";
char *password="telshop-prin-pw";
char *savepw="telshop-prin-pw";
```

- **Define the function prototypes: LookupServer, login, get_binding.**

```
int LookupServer(unsigned char *, trpc_handle_t *);
int login( char *, char *);
rpc_binding_handle_t get_binding (char *, rpc_if_handle_t );
```

- **Set up and define the input area to the IMS transactional RPC source program.**

  The input parameters `stocknum` and `qty` are defined in **mercha.idl**.

```
typedef struct {
                 short int  lll;
                 short int   zzz;
                 char  trxn[7];  /*length of trancode MERCHQ1 */
                 char  txnfill[1];
                 idl_long_int stocknum;
                 idl_long_int qty;
               } ims_in_area;
```

- **Set up and define the output areas from the IMS transactional RPC source program.**

  The output parameters `returncd` and `in_stock` are defined in **mercha.idl**.

```
typedef struct {
                 short  ll;
                 short  zz;
                 idl_long_int  returncd;
                 idl_long_int  in_stock;
               } ims_query_out_area, ims_out_area;

typedef struct {
                 short  ll;
                 short  zz;
                 idl_long_int  returncd;
               } ims_order_out_area;
```

- **Define variables.**

  In this example, the CDS name is hardcoded as the variable `merchServer`. It is used to locate the remote server.

```
    rpc_binding_handle_t     rpcHandle;
    trpc_status_t            trpc_status;
    trpc_handle_t            trpc_handle;
    handle_t                 h;
    unsigned long            st;
    int                      error_inq_st;

    idl_char                 *string_binding, *hostid, *endpoint;
    static idl_char          nil_string[] = "";
    unsigned char*           merchServer="/.:/encina/examples/merch-entry";
```

- **Define code used to communicate error status back to the original client.**

This program uses the following mechanism to communicate back to the client who needs to have a matching code to interpret the error. These values are also coded in the **asecnivt.c** client application. You can use your own mechanism.

```
enum  ims_prog_codes {                /* codes used to communicate */
        RC_OK,                        /* error status back to      */
        ETRAN_ENV_INIT_ERROR,         /* original client           */
        ETRAN_BEG_ERROR,
        CANNOT_OBTAIN_BINDING,
        CANNOT_DCE_LOGIN,
        TRPC_NOT_SUCCESSFUL
       };
```

- **Entry point to program MERCHI1.**

```
main()
{
```

- **IMS C program definitions.**

```
#define io_pcb  ((IO_PCB_TYPE *) (_pcblist[0]))
#define alt_pcb (_pcblist[1])

  static const char func_GU[4] = "GU  ";
  static const char func_ISRT[4] = "ISRT";

  ims_in_area   in_area;
  ims_in_area  *in_area_ptr;

  ims_out_area   out_area;
  ims_out_area  *out_area_ptr;

  char msg_seg_io_area2[100];
  char alt_msg_seg_out[100];
```

- **Application specific definitions.**

```
  long                rc;
  int                 etran_status;
  ecni_error_info_t   ecni_error_info;
```

- **Initialize variables.**

```
  printf("Starting Transaction Merchi1\n");

  out_area.zz = 0;
  out_area.ll = sizeof(ims_out_area);    /* Set up size of output message */
  out_area.returncd = RC_OK;
  in_area_ptr = &in_area;
```

- **Initialize the Encina environment once using the etran_Env_Init API.**

  You must do this before making any transactional RPC calls.

  If you want this program to stay resident across invocations of the transaction (that is, wait for input), you should do this when the transaction program is first called.

```
printf("Calling etran_Env_Init \n");
etran_status = etran_Env_Init(&ecni_error_info);
if (etran_status) {
  printf("etran_Env_Init returns %x\n",etran_status);
  printf("Etran_Env_Init Failure: \n");
  printf("          Failing routine = %s\n",
        ecni_error_info.failing_service_name);
  printf("          Failing code = %d\n",
        ecni_error_info.failing_service_code);
  out_area.returncd = ETRAN_ENV_INIT_ERROR;
   }
```

- **Get a DCE identity.**

  This example program issues one DCE login and issues it outside the Get Unique loop. By doing this, each transactional RPC request will run with the same DCE credential. This program calls a separate C program (**login.c**) to perform the login. See the *z/OS DCE Application Development Guide: Core Components* for more information on DCE security APIs.

  If this is a long running transaction, you must include code to handle the expiration of the login context.

```
else {
printf("Doing a dce_login, rc=%d\n",rc);
strcpy(password,savepw);
rc=login(principal,password);
if (rc) {
   printf("Cannot DCE Login .. Return code = %x\n",rc);
   out_area.returncd = CANNOT_DCE_LOGIN;
   }
```

- **Get a binding handle to base_merchServer using the subroutine LookupServer.**

```
   else {

  printf("Binding to server %s\n",merchServer);
  rc=LookupServer(merchServer, &trpc_handle);
  if (rc){
     printf("cannot bind to server, rc = %x\n", rc );
     out_area.returncd = CANNOT_OBTAIN_BINDING;
     }
   }
 }
```

- **Read each IMS input message.**

  The Get Unique loop iteratively reads each input message and uses IMS DL/1 to process the message queue.

  The `in_area.trxn` field is set to the IMS transaction code of the incoming transaction unless an error is detected, for example, an end-of-queue error. If any error occurs, the field is set to NOTRANS.

  This loop ends if an error occurs during the processing of the transaction or if a DCE exception is raised.

```
TRY
 while (1){   /* do forever */
 rc =CTDLI(func_GU, io_pcb, in_area_ptr);
 printf("CTDLI ret code = %d \n", rc);
 if (rc) {
     strcpy(in_area.trxn,"NOTRANS");
     break;
     }
```

- **Determine the transaction code of the current request.**

  Both `MERCHQ1` (query) and `MERCHO1` (order) transactions are handled by this program. Set up the output area accordingly.

```
if (0==memcmp(in_area.trxn,"MERCHQ1",7))
   out_area.ll = sizeof(ims_query_out_area);
else
   out_area.ll = sizeof(ims_order_out_area);
if (RC_OK != out_area.returncd)
   break;  /* do not process if previous error */
```

- **Activate an Encina transaction.**

```
printf("Calling etran_Begin \n");
etran_status=etran_Begin(&ecni_error_info);
if (etran_status){
   printf("etran_Begin fails with rc = %x\n", etran_status);
   printf("            Failing routine = %s\n",
         ecni_error_info.failing_service_name);
   printf("             Failing code = %d\n",
         ecni_error_info.failing_service_code);
   out_area.returncd = ETRAN_BEG_ERROR;
   break;
   }
```

- **Call the base_merchandise server using transactional RPC.**

  If an error occurs during processing, a DCE exception may be raised. The program branches to `CATCH_ALL`. Refer to the *z/OS DCE Application Development Reference* for information on DCE exception handling and the use of the TRY/CATCH routines. If an Encina error occurs that results in an abort, no IMS work can be made permanent. `merchandise_QueryItem` and `merchandise_OrderItem` are the function names defined in the **merchandise.tidl** file.

```
if (0==memcmp(in_area.trxn,"MERCHQ1",7)){
   ndr_long_int numAvailable=0;
   printf("About to Query Item %d\n",in_area.stocknum);
   merchandise_QueryItem(trpc_handle, in_area.stocknum,
        &out_area.in_stock);
   }
   else {  /* in_area.trxn="MERCHO1" */
     printf("About to Order %d of Item %d\n",in_area.qty,
             in_area.stocknum);
     merchandise_OrderItem(trpc_handle, in_area.stocknum,
             in_area.qty);
     }
```

- **Returns the results to the caller.**

```
rc = CTDLI(func_ISRT, io_pcb, &out_area);
printf("insert ended with rc = %d.\n",rc);
printf("alt_msg_seg_out is %50.50s.\n",alt_msg_seg_out);
```

- **Processing is complete for this IMS input message.**

All work, including transactional RPCs, associated with this input message will be resolved.

```
   }   /* while (1) */
```

- **Cleanup the transactional RPC environment.**

  This point in processing has been reached because either there are no additional input messages or an error was detected while processing the current transaction. If an error was detected while processing a valid input message, insert a response into the message queue.

  Use `etran_Env_Term` to free storage used by IMS transactional RPC processing.

```
   etran_status = etran_Env_Term(&ecni_error_info);

   if (etran_status){
      printf("etran_Env_Term fails with rc = %x\n", etran_status);
      printf("            Failing routine = %s\n",
            ecni_error_info.failing_service_name);
      printf("             Failing code = %d\n",
            ecni_error_info.failing_service_code);
     }

   /* Ensure results are passed back to the client even in the */
   /* event of failure.                                        */

   if (0!=memcmp(in_area.trxn,"NOTRANS",7)){
      printf("insert message with retcd into queue\n");
      rc = CTDLI(func_ISRT, io_pcb, &out_area);
      }

      return(0);
```

- **Handle exceptions caught from a transactional RPC.**

  The program passes control to the `CATCH_ALL` clause if a DCE exception or an Encina exception is raised during IMS transaction RPC processing.

  The environment is cleaned up and a response is sent to the message queue.

```
   CATCH_ALL {
      printf("TRPC error detected\n");
      out_area.returncd = TRPC_NOT_SUCCESSFUL;
      printf("insert message with retcd into queue\n");
      rc = CTDLI(func_ISRT, io_pcb, &out_area);
      etran_status = etran_Env_Term(&ecni_error_info);
      }
   ENDTRY
}     /* main */
```

- **Find a server.**

  The LookupServer routine constructs and returns a transactional handle for a server with the given name. If a handle is found, it is returned. If no handle is found, the program exits. You may use either Encina `trdce_Binding_Import` or z/OS DCE APIs to import the RPC handle to remote servers.

```
int LookupServer(serverName, trpc_handleP)
  unsigned char *serverName;
  trpc_handle_t* trpc_handleP;
{
    unsigned32 status;

    /* Get an RPC handle from the directory service */
```

```
    trdce_BindingImport((unsigned_char_t *) serverName, &rpcHandle, &status);
    printf("LookupServer: trdce_Binding_Import Status=%x\n",status);
    if (0 != status){
       printf("Returning, no Binding Handle\n");
       return(CANNOT_OBTAIN_BINDING);
       }

  /* You must use trpc_ConsBinding to create a transactional RPC handle.     */
  /* Found a valid handle -- use it to construct a transactional RPC handle. */
    status = trpc_ConsBinding(rpcHandle, TRAN_APPL_ID_NULL, TRAN_ADDRESS_NULL,
         TRUE, &trpc_handle);
    if (0 != status){
       printf("Can't construct Binding Handle\n");
       return(CANNOT_OBTAIN_BINDING);
       }
   return(0);
}
```

Chapter 4. Building an IMS Transactional RPC Application

This chapter describes what you must do to build an IMS transactional RPC application (the shaded area in Figure 6). The steps and examples that follow use the names of the sample files shipped with the product. When working through the steps, remember to substitute the sample file names with the actual names of your files—namely, substitute **merchandise** with the name of your **.tidl** file and substitute **merch1i** with the name of your IMS transactional RPC source program. These sample files are shipped with the product and located in `/usr/lpp/encina/example/ecnims/imstran`.

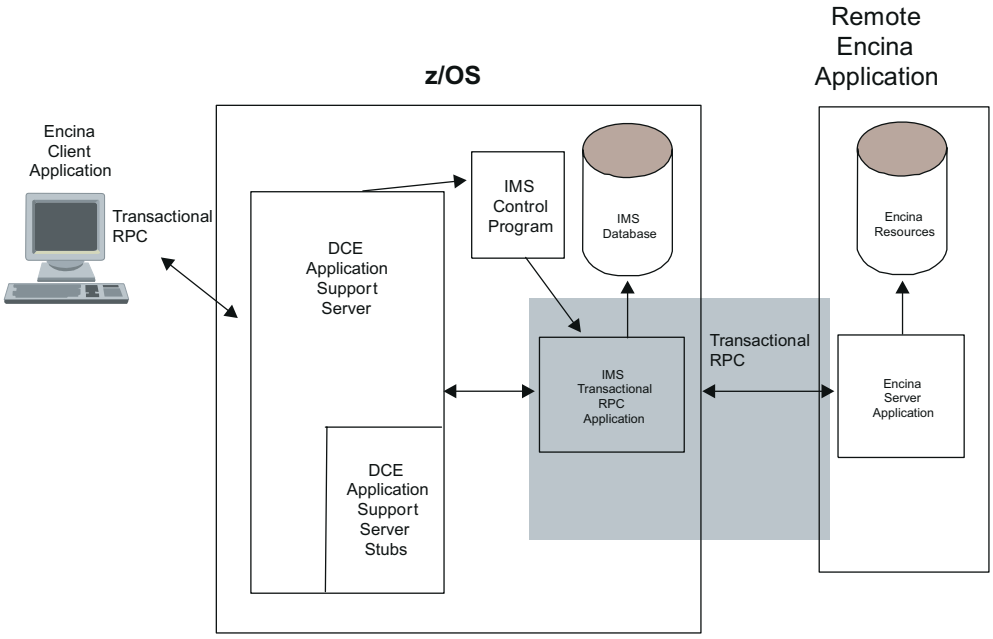

Figure 6. IMS Transactional RPC Application Component of Your Transactional, Distributed, Client/Server Application

To build an IMS transactional RPC application, you must complete the following steps (see Figure 7 on page 24 for an illustration of these steps):

Step 1. Preprocess the transactional interface definition language (TIDL) file.

Step 2. Compile the IDL file generated from the TIDL preprocessing.

Step 3. Compile the IMS transactional RPC source program

Step 4. Compile the C programs generated by IDL and TIDL.

Step 5. Link edit the IMS transactional RPC application.

To build your application, you can use UNIX System Services commands or you can automate these commands using a Makefile similar to the Makefile that is shipped with the product in the `/usr/lpp/encina/example/ecnims/imstran` directory. “Using Commands to Build Your IMS Transactional RPC Application” on page 25 describes the build process using commands. “Using a Makefile to Build an IMS Transactional RPC Application” on page 27 describes our sample Makefile and the values you will need to modify.

We recommend you use a Makefile to build your application; however, you should review “Using Commands to Build Your IMS Transactional RPC Application” on page 25 to understand the commands the Makefile generates. You can edit the Makefile and add your system specific values and then run the Makefile.

You must use UNIX System Services to complete Step 1. However, you can use JCL to perform the IDL compile, C compiles, and link edit described in Step 2 through Step 5.

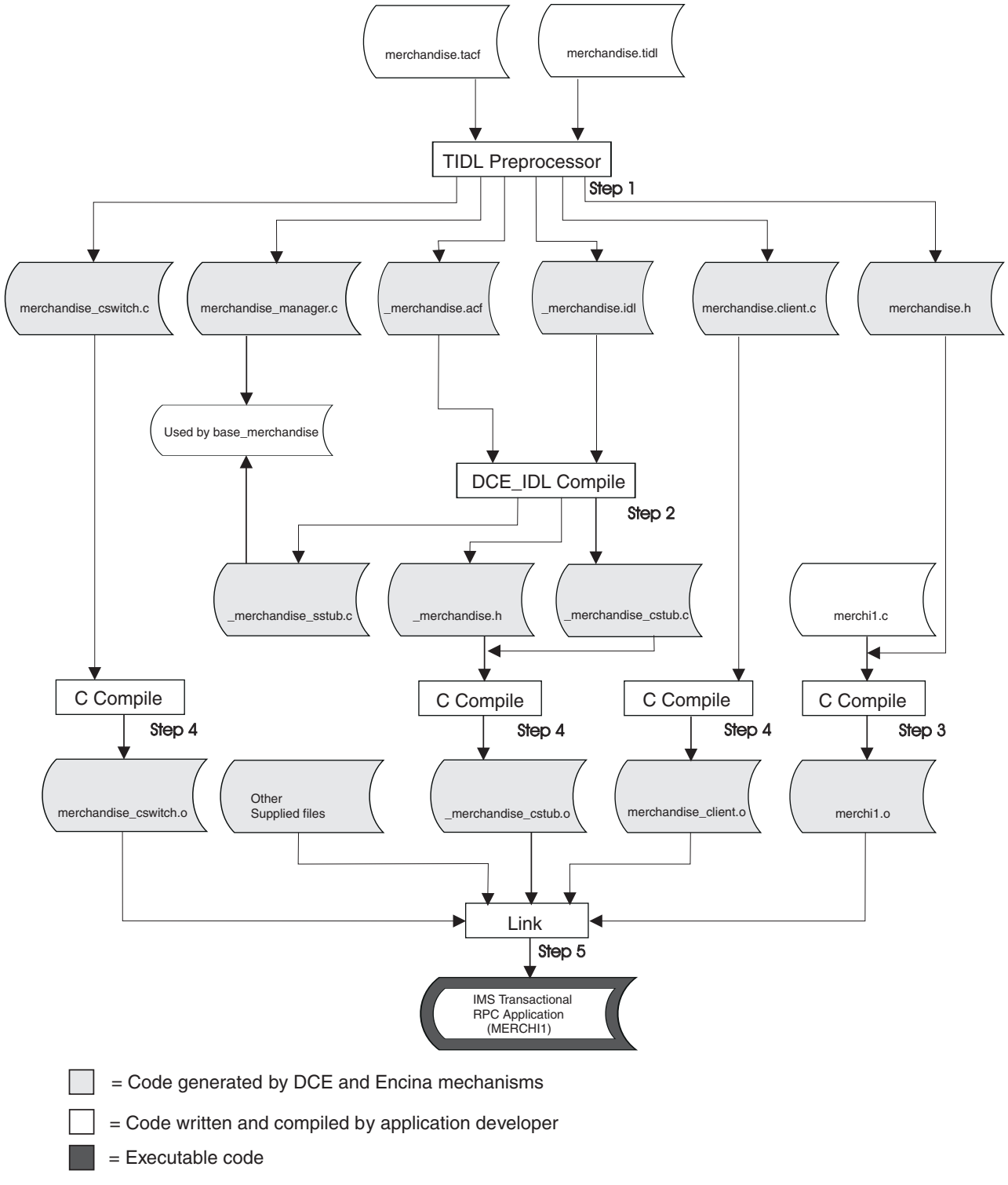

Figure 7. Code, Compile, and Link Steps with Input and Output Files for the Client Application

Using Commands to Build Your IMS Transactional RPC Application

Reminder!

When working through these steps, you must substitute **merchandise**. with the name of your **.tidl** file and substitute **merchi1** with the name of your IMS transactional RPC source program.

Step 1. Preprocess the TIDL File

You have defined the interfaces in the TIDL file; now you must process this file. If you have not created your TIDL file, go to "Defining the Transactional Interface" on page 12.

To preprocess the **.tidl** file, issue the following command:

```
/usr/lpp/encina/bin/tidl -I/usr/lpp/encina/include -no_cpp merchandise.tidl
```

TIDL generates several files by default when it compiles a **.tidl** file (see Step 1 in Figure 7 on page 24). The default files TIDL produces include an IDL file, a header file, and three stub files. The following is a list of the files TIDL generates using **merchandise.tidl** as the **.tidl** file.

merchandise_client.c
: This file contains the shadow client stubs that TIDL produces. It must be compiled and linked with client programs. It contains code that calls the TRPC runtime and the changed operations.

merchandise_cswtch.c
: This file must usually be compiled and linked with the client programs. If an application that uses TIDL wishes to be both a client and a server of an interface, it must not link with this file. Such an application must make the RPC through the entry point vector initialized in **merchandise_client.c**.

_merchandise.idl
: The name of the output interface definition file is **_merchandise.idl**. TIDL also prefixes an underscore (_) to each operation name in the interface definition file. Each operation in this file also has some additional parameters for transmitting and receiving transaction service data and callback data.

merchandise.h
: The client and server programs must include this file instead of the header file IDL produces. It includes the header file IDL produces (**_merchandise.h** in the example).

merchandise_manager.c
: This file contains the shadow manager stubs that TIDL produces. It must be compiled and linked with the server.

_merchandise.acf
: This is an optional auxiliary file. If you used **merchandise.tacf** as input to the tidl preprocessor, TIDL creates a **_merchandise.acf** file.

Refer to *z/OS Encina Toolkit Executive Guide and Reference* for details on the format of the **tidl** command.

Step 2. Compile the IDL File

The **tidl** command in the previous step generated an IDL interface definition file, **_merchandise.idl**. Using the DCE IDL compiler to compile **_merchandise.idl**, use the following command:

```
/usr/lpp/dce/bin/idl -I/usr/lpp/encina/include -I/usr/lpp/dce/share/include \
-cc_opt'-Dunix ' -keep c_source -no_mepv _merchandise.idl
```

This command produces the following three files (see Step 2 in Figure 7 on page 24):

_merchandise_cstub.c
: This file contains the client stubs that marshall the input parameters and unmarshall the output parameters. It is compiled and linked with the client programs.

_merchandise.h
: This file is expected to be included in the client and the server programs when using raw DCE RPC. The header file TIDL produces, **merchandise.h** (in this example), automatically includes this file.

_merchandise_sstub.c
: This file contains the server side stubs that unmarshall the in parameters and marshall the out parameters. It is compiled and linked with the server programs.

Refer to *z/OS Encina Toolkit Executive Guide and Reference* for more information about IDL.

Step 3. Compile Your IMS Transactional RPC Source Program

To compile your IMS transactional RPC source program (see Step 3 in Figure 7 on page 24), use the following command:

Note: Using our sample files you must compile two source programs: **merchi1.c** and **login.c**. **login.c** is a subroutine used by **merchi1.c**.

```
/usr/lpp/encina/etc/c89_encina -g \
-Dunix -D_ALL_SOURCE -DMVS -D_DCE_THREADS -D_ECN_IMS \
-W0,'NOMAR,NOSEQ,DLL,LANGLVL(EXTENDED)' \
-I.  -I/usr/lpp/encina/include \
-I/usr/include/dce  -I"//'CEE.SCEEH.H'"  -I"//'CEE.SCEEH.SYS.H'" \
-c merchi1.c
```

The following is the compile command for the sample program **login.c**:

```
/usr/lpp/encina/etc/c89_encina -g \
-Dunix -D_ALL_SOURCE -DMVS -D_DCE_THREADS -D_ECN_IMS \
-W0,'NOMAR,NOSEQ,DLL,LANGLVL(EXTENDED)' \
-I.  -I/usr/lpp/encina/include \
-I/usr/include/dce  -I"//'CEE.SCEEH.H'"  -I"//'CEE.SCEEH.SYS.H'" \
-c login.c
```

Step 4. Compile the Generated C Program

The previous TIDL preprocess step and the IDL compile step generated three C programs:

- The IDL-generated client stub (**_merchandise_cstub.c**)
- The client switch file (**merchandise_cswitch.c**)
- The client shadow file (**merchandise_client.c**)

Now, you must compile these three programs (see Step 4 in Figure 7 on page 24):

```
/usr/lpp/encina/etc/c89_encina -g \
-Dunix -D_ALL_SOURCE -DMVS -D_DCE_THREADS -D_ECN_IMS \
-W0,'NOMAR,NOSEQ,DLL,LANGLVL(EXTENDED)' \
-I/usr/lpp/encina/include \
-I/usr/include/dce  -I"//'CEE.SCEEH.H'"  -I"//'CEE.SCEEH.SYS.H'" \
-c _merchandise_cstub.c

/usr/lpp/encina/etc/c89_encina -g \
-Dunix -D_ALL_SOURCE -DMVS -D_DCE_THREADS -D_ECN_IMS \
-W0,'NOMAR,NOSEQ,DLL,LANGLVL(EXTENDED)' \
-I/usr/lpp/encina/include \
-I/usr/include/dce  -I"//'CEE.SCEEH.H'"  -I"//'CEE.SCEEH.SYS.H'" \
-c merchandise_cswtch.c

/usr/lpp/encina/etc/c89_encina -g
-Dunix -D_ALL_SOURCE -DMVS -D_DCE_THREADS -D_ECN_IMS
-W0,'NOMAR,NOSEQ,DLL,LANGLVL(EXTENDED)'
-I/usr/lpp/encina/include
-I/usr/include/dce  -I"//'CEE.SCEEH.H'"  -I"//'CEE.SCEEH.SYS.H'"
-c merchandise_client.c
```

For information about compiling, see the documentation of the C compiler for your operating system.

Step 5. Build the IMS Transactional RPC Application

To create the IMS transactional RPC application, link-edit the object files you just created. (See Step 5 in Figure 7 on page 24.) In our example, the object files are: **merchi1.o**, **_merchandise_cstub.o**, **merchandise_cswitch.o**, **merchandise_client.o**, and **login.o**.

```
/usr/lpp/encina/etc/c89_encina -Wl,'AMODE=31,RMODE=ANY,' -Wl,'AC=1' \
-Wl,DLL -L/usr/lpp/encina/lib/ \
-L/usr/lpp/dce/lib  -L/usr/lib \
/usr/lpp/encina/lib/libEncina.x  /usr/lpp/encina/lib/ECNIDLL.x \
/usr/lpp/dce/lib/EUVPDLL.x -o"//'YOUR.PGMLIB(merchi1)'" \
_merchandise_cstub.o  merchandise_cswtch.o merchandise_client.o \
login.o  merchi1.o -l"//'IMS6QSYS.RESLIB'" -lecni -ldce
```

Using a Makefile to Build an IMS Transactional RPC Application

Before using a Makefile to automate the build process, you must first edit the Makefile shipped with the product and add your system specific values to the file. Then, you can run the Makefile.

In addition to changing the name of **.tidl** file and the IMS transactional RPC source program where appropriate, you may require additional include paths and compiler and link-edit options. See the *z/OS UNIX System Services Command Reference* for details on creating a Makefile.

Step 1. Modify the Makefile

Review and modify the following elements of the Makefile:

- Set the name of the PDS into which the IMS transactional RPC application is to be built. You can obtain this dataset name from your IMS administrator.
- Set the dataset name where your IMS RESLIB dataset is placed.
- Define installation directories.

DCE_PATH
: points to the directory in which DCE is installed

ENCINA_DIR
: points to the directory in which the Encina Toolkit Executive is installed

EXAMPLE_DIR
: points to the directory in which the examples and samples are installed

ECNI_PATH
: points to the directory in which the Encina Toolkit Executive is installed

CEEH and CEESYSH
: points to the directory in which the Communication Execution Environment headers are installed.

- Set the include paths.

  Include files are text files specifying structure, variable, and macro definitions used in applications written in the C programming language. The include files for the Encina Toolkit are organized hierarchically under the **include** subdirectory of the Encina Toolkit Executive installation directory.

  The C compiler's **-I** option specifies additional directories to search for include files. The name of the additional directory to search must follow this option.

  Toolkit Executive applications that use any of the services that the DCE modules provide (such as DCE RPCs, the DCE Directory Service, and the DCE Time Service) must search additional directories to find all of the include files required for these modules. For example, applications using DCE RPCs require additional include files for that module when compiling the source files produced by IDL and Transarc's **TIDL** programs. For example:

  ```
  -I$(ENCINA_DIR)/include
  ```

  The include files for z/OS DCE applications are located in the **/usr/lpp/dce/share/include** directory. To add the z/OS DCE directory to the standard list of directories searched for include files during compilation, add the following Makefile statement:

  ```
  -I$(DCE_PATH)/share/include
  ```

- Set flags for the IDL and C compilers.

  CFLAGS by default, CFLAGS flags are used by c89 compiler.

- Set Toolkit and DCE libraries.

  Note: In z/OS, DCE and Encina libraries are implemented as DLLs.

  In addition to the Encina and DCE libraries, there are two additional files needed to build an IMS transactional RPC application—**libecni.a** and **ecnidll.x**.

Step 2. Run the Makefile

Now, to build your IMS transactional RPC application using the Makefile, issue the following command:

```
make -f Makefile
```

Building the Server

The previous steps generated source files that you will need when building the server side of the application (see Figure 7 on page 24). In our example, these files are:

```
merchandise_manager.c
_merchandise_sstub.c
merchandise.h
_merchandise.h
```

For information on building the server application, see the *z/OS Encina Toolkit Executive Guide and Reference*.

Chapter 5. Managing Transactions

You can manually commit or abort a transaction or determine the state of a transaction using the **tkadmin** command from a non-z/OS platform. AS IMS performs all Encina transactional processing for the IMS transactional RPC application. The application ID of AS IMS and any servers contacted by your IMS transactional RPC application appear as participants in the Encina transaction, but your IMS transactional RPC application does not appear as a participant. Therefore, when using the **tkadmin** command, you must use the server name of AS IMS to perform administrative tasks on behalf of your IMS transactional RPC application.

For details on the **tkadmin** command, see the Administrator's Guide for the Encina Toolkit for your system. For information describing how to manage AS using the **tkadmin** command, refer to “Managing Transactions Called with Transactional RPCs” in the *z/OS DCE Application Support Configuration and Administration Guide*.

Appendix A. IMS Transactional RPC Programming Interfaces

When you write IMS applications that perform Encina transactional RPCs, you must use the following C language IMS transactional RPC programming interfaces:

etran_Begin

etran_Env_Init

etran_Env_Term

etran_Begin

Makes the current IMS transaction part of an Encina transaction unit. **etran_Begin** ensures that all updates to Encina recoverable resources performed as a result of the IMS transaction are coordinated with all other recoverable participants during two-phase commit processing of the originating Encina request.

When you code etran_Begin, you do not need to explicitly delimit the end of an Encina transaction. The Encina commit scope is the same as the IMS transaction itself. The Encina client program is expected to drive commit processing.

The IMS transaction must issue **etran_Begin** before making any Encina calls for the current unit of recovery. You must initialize the IMS transactional RPC environment using **etran_Env_Init** before using **etran_Begin**.

Format

int etran_Begin(*ecni_error_info_t* ***)**

Parameters

*ecni_error_info_t **

points to a structure of type `ecni_error_info_t`, which is defined in **ecnims/ecnims.h**. *ecni_error_info_t* returns the following error information:

failing_service_name

is a 1-32 byte character string that indicates what routine generated the error. This may be an IMS transactional RPC API or a secondary service called by this routine.

failing_service_code

is the return code of the failing service. The following is a list of the return values if the failing service is an IMS transactional RPC API:

ETRAN_BEGIN_ALREADY_DONE
: This IMS transaction already called **etran_Begin** for this unit of recovery.

ETRAN_INIT_NOTDONE
: The IMS transactional RPC environment has not been initialized.

ETRAN_BEGIN_NO_TRPC_IN_PROGRESS
: This IMS transaction has not been initiated as part of an active Encina transaction.

INSUFFICIENT_STORAGE
: An error occurred while attempting to allocate storage.

ECNI_INTERNAL_ERROR
: An unexpected condition was detected. Contact your IBM support representative.

Results

Return Value	Meaning
SUCCESS	
ERROR	The call was unsuccessful.
SEVERE	The service detected an unexpected state. Contact your IBM service representative.

Related Topics

etran_Env_Init

etran_Env_Term

etran_Env_Init

Sets up the Encina environment in the process where the IMS transaction is running. You must issue this before any transactional RPCs are initiated. You can issue this only once per IMS transactional RPC program. **etran_Env_Init** enables your IMS transactional RPC source program to:

- Issue IMS transactional RPC APIs
- Issue **trdce** calls
- Use transactional RPCs using interfaces processed by the -tidl processor
- Construct trpc binding handles using the **trpc_ConsBinding** function.

Other z/OS Encina Toolkit interfaces are not supported.

Format

int etran_Env_Init(*ecni_error_info_t* ***)**

Parameters

*ecni_error_info_t **

points to a structure of type `ecni_error_info_t`, which is defined in **ecnims/ecnims.h**. *ecni_error_info_t* returns the following error information:

failing_service_name

is a 1-32 byte character string that indicates what routine generated the error. This may be an IMS transactional RPC API or a secondary service called by this routine.

failing_service_code

is the return code of the failing service. The following is a list of the return values if the failing service is an IMS transactional RPC API:

ETRAN_INIT_ALREADY_DONE

You can call **etran_Env_Init** only once per IMS transactional RPC program, and you have already called **etran_Env_Init**.

INSUFFICIENT_STORAGE

An error occurred while attempting to allocate storage.

ECNI_INTERNAL_ERROR

An unexpected condition was detected. Contact your IBM support representative.

Results

Return Value	Meaning
SUCCESS	
ERROR	The call was unsuccessful.
SEVERE	The service detected an unexpected state. Contact your IBM service representative.

Related Topics

etran_Begin

etran_Env_Term

etran_Env_Term

Deallocates storage obtained by IMS transactional RPC processing. IMS transactional RPC source programs may issue **etran_Env_Term** to delete storage. You may not reinitialize the Encina environment.

Format

int etran_Env_Term(*ecni_error_info_t* ***)**

Parameters

ecni_error_info_t *

points to a structure of type `ecni_error_info_t`, which is defined in **ecnims/ecnims.h**. *ecni_error_info_t* returns the following error information:

failing_service_name

is a 1-32 byte character string that indicates what routine generated the error. This may be an IMS transactional RPC API or a secondary service called by this routine.

failing_service_code

is the return code of the failing service. The following is a list of the return values if the failing service is an IMS transactional RPC API:

ETRAN_INIT_NOTDONE

The IMS transactional RPC environment has not been initialized.

ECNI_INTERNAL_ERROR

Results

Return Value	Meaning
SUCCESS	
ERROR	The call was unsuccessful.
SEVERE	The service detected an unexpected state. Contact your IBM service representative.

Related Topics

etran_Env_Init

etran_Begin

Appendix B. 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 PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some 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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM 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
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
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 information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

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.

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

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

BookManager	CICS	CICS/ESA
IBM	IBMLink	IMS
IMS/ESA	Language Environment	Library Reader
MVS/ESA	OS/390	RACF
Resource Link	SecureWay	VTAM
z/OS		

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Programming Interface Information

z/OS Encina Transactional RPC Support for IMS documents intended Programming Interfaces that allow the customer to write IMS applications that perform transactional RPCs.

Glossary

This glossary defines technical terms and abbreviations used in z/OS DCE documentation. If you do not find the term you are looking for, refer to the index of the appropriate z/OS DCE manual or view the *IBM Glossary of Computing Terms*, located at:

http://www.ibm.com/ibm/terminology

This glossary includes terms and definitions from:

- *IBM Dictionary of Computing*, SC20-1699.
- *Information Technology—Portable Operating System Interface (POSIX),* from the POSIX series of standards for applications and user interfaces to open systems, copyrighted by the Institute of Electrical and Electronics Engineers (IEEE).
- *American National Standard Dictionary for Information Systems,* ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- *Information Technology Vocabulary,* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1.SC1).
- *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the International Telecommunication Union, Geneva, 1978.
- Open Software Foundation (OSF).

The following abbreviations indicate terms that are related to a particular DCE service:

CDS	Cell Directory Service
CICS/ESA®	Customer Information Control System/ESA
DTS	Distributed Time Service
GDS	Global Directory Service
IMS/ESA®	Information Management System/ESA
RPC	Remote Procedure Call
Security	Security Service
Threads	Threads Service
XDS	X/Open Directory Services
XOM	X/Open OSI-Abstract-Data Manipulation

A

abort. To fail to commit. Any changes made by an Encina transaction that is aborted, for whatever reason, are undone. Once an Encina transaction is undone (rolled back), no evidence that it was ever attempted remains outside of records in the transaction processing system's log. See also *commit*.

access control list (ACL). (1) GDS: Specifies the users with their access rights to an object. (2) Security: Data that controls access to a protected object. An ACL specifies the privilege attributes needed to access the object and the permissions that may be granted, to the protected object, to principals that possess such privilege attributes.

access right. Synonym for *permission*.

accessible. Pertaining to an object whose client possesses a valid designator or handle.

account. Data in the Registry database that allows a principal to log in. An account is a registry object that relates to a principal.

ACL. Access control list.

adapter. Synonym for *attachment facility*.

address. An unambiguous name, label, or number that identifies the location of a particular entity or service. See *presentation address*.

Advanced Program-to-Program Communications (APPC). An implementation of the SNA/SDLC LU6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

APF. Authorized program facility.

API. Application program interface.

APPC. Advanced Program-to-Program Communications.

application identifier. A unique identifier used to identify an application in the transactional RPCs sent in a distributed environment.

application program interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to

use specific data or functions of the operating system or the licensed program.

Application Support server. Refers to the server for z/OS DCE Application Support. The Application Support server allows a client program to access CICS or IMS.

application thread. A thread of execution created and managed by application code. See *client application thread*, *local application thread*, *RPC thread*, and *server application thread*.

applId. Application identifier. A unique identifier used to identify an application in the transactional RPCs sent in a distributed environment.

architecture. (1) The organizational structure of a computer system, including the interrelationships among its hardware and software. (2) The logical structure and operating principles of a computer network. The operating principles of a network include those of services, functions, and protocols.

ASPREFX. The high-level qualifier of the MVS data set where your Application Support files are installed.

asynchronous. Without a regular time relationship; unexpected or unpredictable with respect to the running of program instructions.

attachment facility. Application Support server: Refers to the CICS adapter and the IMS adapter. Synonymous with *adapter*.

attribute. (1) RPC: An Interface Definition Language (IDL) or attribute configuration file (ACF) that conveys information about an interface, type, field, parameter, or operation. (2) DTS: A qualifier used with DTS commands. DTS has four attribute categories: characteristics, counters, identifiers, and status. (3) XDS: Information of a particular type concerning an object and appearing in an entry that describes the object in the directory information base (DIB). It denotes the attribute's type and a sequence of one or more attribute values, each accompanied by an integer denoting the value's syntax.

attribute syntax. GDS: A definition of the set of values that an attribute may assume. Attribute syntax includes the data type, in ASN.1, and usually one or more matching rules by which values may be compared.

attribute type. (1) XDS: The component of an attribute that indicates the type of information given by that attribute. Because it is an object identifier, it is unique among other attribute types. (2) XOM: Any of various categories into which the client dynamically groups values on the basis of their semantics. It is an integer unique only within the package.

attribute value. XDS, XOM: A particular instance of the type of information indicated by an attribute type.

authentication. In computer security, a method used to verify the identity of a principal.

Authentication Service. One of three services provided by the Security Service: it verifies principals according to a specified authentication protocol. The other Security services are the Privilege Service and the Registry Service.

authorization. (1) The determination of a principal's permissions with respect to a protected object. (2) The approval of a permission sought by a principal with respect to a protected object.

authorized program facility (APF). An MVS facility that permits identification of programs authorized to use restricted functions.

B

binding. RPC: A relationship between a client and a server involved in a remote procedure call.

binding handle. RPC: A reference to a binding. See *binding information*.

binding information. RPC: Information about one or more potential bindings, including an RPC protocol sequence, a network address, an endpoint, at least one transfer syntax, and an RPC protocol version number. See *binding*. See also *endpoint*, *network address*, *RPC protocol*, *RPC protocol sequence*, and *transfer syntax*.

broadcast. A notification sent to all members within an arbitrary grouping such as nodes in a network or threads in a process. See also *signal*.

C

cache. (1) CDS: The information that a CDS clerk stores locally to optimize name lookups. The cache contains attribute values resulting from previous lookups, as well as information about other clearinghouses and namespaces. (2) Security: Contains the credentials of a principal after the DCE login. (3) GDS: See *DUA cache*.

call thread. RPC: A thread created by an RPC server's runtime to run remote procedures. When engaged by a remote procedure call, a call thread temporarily forms part of the RPC thread of the call. See *application thread* and *RPC thread*.

cancel. (1) Threads: A mechanism by which a thread informs either itself or another thread to stop the thread

as soon as possible. If a cancel arrives during an important operation, the canceled thread may continue until it can end the thread in a controlled manner. (2) RPC: A mechanism by which a client thread notifies a server thread (the canceled thread) to end the thread as soon as possible. See also *thread*.

CDS. Cell Directory Service.

CDS clerk. The software that provides an interface between client applications and CDS servers.

CDS control program (CDSCP). A command interface that CDS administrators use to control CDS servers and clerks and manage the name space and its contents. See also *manager*.

CDSCP. CDS control program.

cell. The basic unit of operation in the distributed computing environment. A cell is a group of users, systems, and resources that are grouped around a common purpose and that share common DCE services.

Cell Directory Service (CDS). A DCE component. A distributed replicated database service that stores names and attributes of resources located in a cell. CDS manages a database of information about the resources in a group of machines called a DCE cell.

CICS. Customer Information Control System.

class. A category into which objects are placed on the basis of their purpose and internal structure.

clerk. (1) DTS: A software component that synchronizes the clock for its client system by requesting time values from servers, calculating a new time from the values, and supplying the computed time to client applications. (2) CDS: A software component that receives CDS requests from a client application, ascertains an appropriate CDS server to process the requests, and returns the results of the requests to the client application.

client. A computer or process that accesses the data, services, or resources of another computer or process on the network. Contrast with *server*.

client application thread. RPC: A thread executing client application code that makes one or more remote procedure calls. See *application thread*, *local application thread*, *RPC thread*, and *server application thread*.

client context. RPC: The state within an RPC server generated by a set of remote procedures and maintained across a series of calls for a particular client. See *context handle*. See also *manager*.

client stub. RPC: The surrogate code for an RPC interface that is linked with and called by the client application code. In addition to general operations such as marshalling data, a client stub calls the RPC runtime to perform remote procedure calls and, optionally, to manage bindings. See *server stub*.

code page. (1) A table showing codes assigned to character sets. (2) An assignment of graphic characters and control function meanings to all code points. (3) Arrays of code points representing characters that establish numeric order of characters. [OSF] (4) A particular assignment of hexadecimal identifiers to graphic elements. (5) Synonymous with code set. (6) See also *code point, extended character*.

collapse. CDS: To remove the contents of a directory from the display (close it) using the CDS Browser. To collapse an open directory, double-click on its icon. Double-clicking on a closed directory expands it. Contrast with *expand*.

commit. To make all updates permanent. When an Encina transaction commits, all actions associated with that specific transaction have been written to the log. Even in the event of system problems, those actions are repeated if necessary when the system's recovery mechanism replays the log. See also *abort*.

context handle. RPC: A reference to state (client context) maintained across remote procedure calls by a server on behalf of a client. See *client context*.

control access. CDS: An access right that grants users the ability to change the access control on a name and to perform other powerful management tasks, such as replicate a directory or move a clearinghouse.

copy. GDS, XDS: Either a copy of an entry stored in other DSAs through bilateral agreement or a locally and dynamically stored copy of an entry resulting from a request (a cache copy).

coupling facility. A special logical partition that provides high-speed caching, list processing, and locking functions in a sysplex.

credentials. Security: A general term for privilege attribute data that has been certified by a trusted privilege certification authority.

cross-linking information. In order for z/OS DCE to provide RACF-DCE interoperability and single sign-on to DCE, DCE provides utilities (see **mvsexpt** and **mvsimpt**) to incorporate into RACF the information that associates a z/OS-RACF user ID with a DCE principal's identifying information and the DCE principal's UUID with the corresponding z/OS-RACF user ID. The information is placed in a RACF DCE segment and the RACF general resource class, DCEUUIDS. This is called **cross-linking information** and is what allows

interoperability and single sign-on to work. See also *interoperability* and *single sign-on*.

cross-system coupling facility (XCF). A component of MVS that provides functions to support cooperation between authorized programs running within a sysplex.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

D

daemon. (1) A long-lived process that runs unattended to perform continuous or periodic system-wide functions such as network control. Some daemons are triggered automatically to perform their task; others operate periodically. An example is the **cron** daemon, which periodically performs the tasks listed in the **crontab** file. Many standard dictionaries accept the spelling *demon*. (2) A DCE server process.

DCE. Distributed Computing Environment.

DFS. Distributed File Service.

directory. (1) A logical unit for storing entries under one name (the directory name) in a CDS namespace. Each physical instance of a directory is called a replica. (2) A collection of open systems that cooperates to hold a logical database of information about a set of objects in the real world.

directory ID. Directory identifier.

Directory Service. A DCE component. The Directory Service is a central repository for information about resources in a distributed system. See *Cell Directory Service* and *Global Directory Service*.

directory system. GDS: A system for managing a directory, consisting of one or more DSAs. Each DSA manages part of the DIB.

distributed computing. A type of computing that allows computers with different hardware and software to be combined on a network, to function as a single computer, and to share the task of processing application programs.

Distributed Computing Environment (DCE). A comprehensive, integrated set of services that supports the development, use, and maintenance of distributed applications. DCE is independent of the operating system and network; it provides interoperability and portability across heterogeneous platforms.

Distributed File Service (DFS). A DCE component. DFS joins the local file systems of several file server machines making the files equally available to all DFS client machines. DFS allows users to access and share files stored on a file server anywhere in the network, without having to consider the physical location of the file. Files are part of a single, global name space, so that a user can be found anywhere in the network by means of the same name.

Distributed Time Service (DTS). A DCE component. It provides a way to synchronize the times on different hosts in a distributed system.

DLL. Dynamic link library.

DTS. Distributed Time Service.

DTS entity. DTS: The server or clerk software on a system.

DUA cache. GDS: The part of the DUA that stores information to optimize name lookups. Each cache contains copies of recently accessed object entries as well as information about DSAs in the directory.

dynamic link library (DLL). Binds parts of the executable at load or runtime.

E

EID. An AS IMS identifier consisting of the Encina GTID and TID associated with a given transaction.

element. RPC: Any of the bits of a bit string, the octets of an octet string, or the octets by means of which the characters of a character string are represented.

Encina. A family of software products for building and running large-scale, distributed client-server systems. Encina uses and enhances the facilities DCE provides with transactional semantics.

Encina recovery server. A z/OS UNIX DCE server.

endpoint. RPC: An address of a specific server instance on a host.

endpoint map. RPC: A database local to a node where local RPC servers register binding information associated with their interface identifiers and object identifiers. The endpoint map is maintained by the endpoint map service of the DCE daemon.

endpoint map service. RPC: A service that maintains a system's endpoint map for local RPC servers. When an RPC client makes a remote procedure call using a partially bound binding handle, the endpoint map

service looks up the endpoint of a compatible local server. See *endpoint map*.

entity. (1) CDS: Any manageable element through the CDS namespace. Manageable elements include directories, object entries, servers, replicas, and clerks. The CDS control program (CDSCP) commands are based on directives targeted for specific entities. (2) DTS: See *DTS entity*.

entry. GDS, XDS: The part of the DIB that contains information relating to a single directory object. Each entry consists of directory attributes.

ENV. Environment variable.

environment variable (ENV). A variable included in the current software environment that is available to any called program that requests it.

ephemeral application. An application that does not contain any recoverable data. See *recoverable data*.

exception. (1) An abnormal condition such as an I/O error encountered in processing a data set or a file. (2) One of five types of errors that can occur during a floating-point exception. These are valid operation, overflow, underflow, division by zero, and inexact results. [OSF] (3) Contrast with *interrupt, signal*.

executor thread. See *call thread*.

expand. CDS: To display the contents of (open) a directory using the CDS Browser. A directory that is closed can be expanded by double-clicking on its icon. Double-clicking on an expanded directory collapses it. Contrast with *collapse*.

export. (1) RPC: To place the server binding information associated with an RPC interface or a list of object UUIDs or both into an entry in a name service database. (2) To provide access information for an RPC interface. Contrast with *unexport*.

F

foreign cell. A cell other than the one to which the local machine belongs. A foreign cell and its binding information are stored in either GDS or the Domain Name System (DNS). The act of contacting a foreign cell is called intercell. Contrast with *local cell*.

full name. CDS: The complete specification of a CDS name, including all parent directories in the path from the cell root to the entry being named.

fully bound binding handle. RPC: A server binding handle that contains a complete server address including an endpoint. Contrast with *partially bound binding handle*.

G

GDS. Global Directory Service.

General-Use Programming Interface (GUPI). An interface, with few restrictions, for use in customer-written programs. The majority of programming interfaces are general-use programming interfaces, and are appropriate in a wide variety of application programs. A general-use programming interface requires the knowledge of the externals of the interface and perhaps the externals of related programming interfaces. Knowledge of the detailed design or implementation of the software product is not required.

Global Directory Service (GDS). A DCE component. A distributed replicated directory service that provides a global namespace that connects the local DCE cells into one worldwide hierarchy. DCE users can look up a name outside a local cell with GDS.

global name. A name that is universally meaningful and usable from anywhere in the DCE naming environment. The prefix /... indicates that a name is global.

group. (1) RPC: A name service entry that corresponds to one or more RPC servers that offer common RPC interfaces, RPC objects, or both. A group contains the names of the server entries, other groups, or both that are members of the group. See *NSI group attribute*. (2) Security: Data that associates a named set of principals that can be granted common access rights. See *subject identifier*.

GTID. Encina global transaction identifier (global TID). An identifier that is unique across all servers. This ties a transaction to all of its participating applications across different servers.

GUPI. General-Use Programming Interface.

H

handle. RPC: An opaque reference to information. See *binding handle*, *context handle*, *interface handle*, *name service handle*, and *thread handle*.

home cell. Synonym for *local cell*.

host ID. Synonym for *network address*.

I

IDL. Interface Definition Language.

IDL compiler. RPC: A compiler that processes an RPC interface definition and an optional attribute configuration file (ACF) to generate client and server stubs, and header files. For Application Support, z/OS extends the IDL compiler to allow generation of server stubs for IMS or CICS transactions, COBOL data types, and Encina transactional RPCs. The z/OS IDL compiler option **-tidl** causes z/OS Encina Toolkit Executive TIDL preprocessing to occur transparently to the user, producing a transactional Application Support server stub.

IMS. Information Management System.

IMS control region. An IMS address space that controls the overall flow of information, execution, and data access within a set of IMS-dependent regions.

IMS recovery token. The identifier of a transaction for IMS.

Information Management System (IMS). A database and data communication system capable of managing complex databases and networks in virtual storage.

instance. XOM: An object in the category represented by a class.

interface. RPC: A shared boundary between two or more functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions. See *RPC interface*.

interface definition. RPC: A description of an RPC interface written in the DCE Interface Definition Language (IDL). See *RPC interface*.

Interface Definition Language (IDL). A high-level declarative language that provides syntax for interface definitions.

For Application Support, z/OS extends IDL to support COBOL and C data types, Encina transactional attributes, and extensions to communicate with CICS and IMS.

interface handle. RPC: A reference in code to an interface specification. See *binding handle* and *interface specification*.

interface identifier. RPC: A string containing the interface Universal Unique Identifier (UUID) and major and minor version numbers of a given RPC interface. See *RPC interface*.

interface specification. RPC: An opaque data structure that is generated by the DCE IDL compiler from an interface definition. It contains identifying and descriptive information about an RPC interface. See *interface definition*, *interface handle*, and *RPC interface*.

interface UUID. RPC: The Universal Unique Identifier (UUID) generated for an RPC interface definition using the UUID generator. See *interface definition* and *RPC interface*.

Internet Protocol (IP). In TCP/IP, a protocol that routes data from its source to its destination in an Internet environment. IP provides the interface from the higher level host-to-host protocols to the local network protocols. Addressing at this level is usually from host to host.

interoperability. The capability to communicate, execute programs, or transfer data among various functional units in a way that requires the user to have little or no knowledge of the unique characteristics of those units.

intersystem communication (ISC). IMS/ESA: An extension of the IMS multiple systems coupling feature that permits the connection of IMS to another IMS subsystem, to CICS, or to a user-written subsystem, provided both systems use ISC.

IP. Internet Protocol.

ISC. Intersystem communication.

K

Kerberos. The authentication protocol used to carry out DCE private key authentication. Kerberos was developed at the Massachusetts Institute of Technology.

key. A value used to encrypt and decrypt data.

L

local. (1) Pertaining to a device directly connected to a system without the use of a communication line. (2) Pertaining to devices that have a direct, physical connection. Contrast with *remote*.

local application thread. RPC: An application thread that runs within the confines of one address space on a local system and passes control exclusively among local code segments. See *application thread*, *client application thread*, *RPC thread* and *server application thread*.

local cell. The cell to which the local machine belongs. Synonymous with *home cell*. Contrast with *foreign cell*.

logical unit (LU). A host port through which a user gains access to the services of a network.

lookup context. The context setup by the client to locate compatible binding handles from the name space. Name service interfaces (NSI) are used to setup and free the lookup context.

LU. Logical unit.

M

manager. RPC: A set of remote procedures that implement the operations of an RPC interface and that can be dedicated to a given type of object. See also *object* and *RPC interface*.

mask. (1) A pattern of characters used to control the retention or deletion of portions of another pattern of characters (2) Security: Used to establish maximum permissions that can then be applied to individual ACL entries. (3) GDS: The administration screen interface menus.

message format service (MFS). An editing facility that allows application programs to deal with simple logical messages instead of device-dependent data, thus simplifying the application development process.

mutex. Mutual exclusion. A read/write lock that grants access to only a single thread at any one time. A mutex is often used to ensure that shared variables are always seen by other threads in a consistent way.

mvsexpt. One of two (the other is **mvsimpt**) utilities used to automate much of the administrator's work in creating the cross-linking information for DCE-RACF interoperability. The **mvsexpt** utility creates the cross-linking information in the RACF database from information in the DCE registry. See also *cross-linking information*, *interoperability*, and *single sign-on*.

mvsimpt. One of two (the other is **mvsexpt**) utilities used to automate much of the administrator's work in creating the cross-linking information for DCE-RACF interoperability. The **mvsimpt** utility creates DCE principals from information obtained from the RACF database. See also *cross-linking information*, *interoperability*, and *single sign-on*.

N

name. GDS, CDS: A construct that singles out a particular (directory) object from all other objects. A name must be unambiguous (denote only one object); however, it need not be unique (be the only name that unambiguously denotes the object).

name service handle. RPC: An opaque reference to the context used by the series of next operations called during a specific name service interface (NSI) search or inquiry.

namespace. CDS: A complete set of CDS names that one or more CDS servers look up, manage, and share. These names can include directories, object entries, and soft links.

network. A collection of data processing products connected by communications lines for exchanging information between stations.

network address. An address that identifies a specific host on a network. Synonymous with *host ID*.

Network Data Representation (NDR). RPC: The transfer syntax defined by the Network Computing Architecture. See *transfer syntax*.

network protocol. A communications protocol from the Network Layer of the Open Systems Interconnection (OSI) network architecture, such as the Internet Protocol (IP).

node. (1) An endpoint of a link, or a junction common to two or more links in a network. Nodes can be preprocessors, controllers, or workstations, and they can vary in routing and other functional capabilities. (2) In network topology, the point at an end of a branch. It is usually a physical machine.

NSI binding attribute. RPC: An RPC-defined attribute (NSI attribute) of a name service entry; the binding attribute stores binding information for one or more interface identifiers offered by an RPC server and identifies the entry as an RPC server entry. See *binding information* and *NSI object attribute*. See also *server entry*.

NSI group attribute. RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores the entry names of the members of an RPC group and identifies the entry as an RPC group. See *group*.

NSI object attribute. RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores the object UUIDs of a set of RPC objects. See *object*.

NSI profile attribute. RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores a collection of RPC profile elements and identifies the entry as an RPC profile. See *profile*.

NULL. In the C language, a pointer that does not point to a data object.

O

object. (1) A data structure that implements some feature and has an associated set of operations. (2) RPC: For RPC applications, anything that an RPC server defines and identifies to its clients using an object Universal Unique Identifier (UUID). An RPC object is often a physical computing resource such as a database, directory, device, or processor. Alternatively, an RPC object can be an abstraction that is meaningful to an application, such as a service or the location of a server. See *object UUID*. (3) XDS: Anything in the world of telecommunications and information processing that can be named and for which the directory information base (DIB) contains information. (4) XOM: Any of the complex information objects created, examined, changed, or destroyed by means of the interface.

object entry. CDS: The name of a resource (such as a node, disk, or application) and its associated attributes, as stored by CDS. CDS administrators, client application users, or the client applications themselves can give a resource an object name. CDS supplies some attribute information (such as a creation time stamp) to become part of the object, and the client application may supply more information for CDS to store as other attributes. See *entry*.

object identifier (OID). A value (distinguishable from all other such values) that is associated with an information object. It is formally defined in the CCITT X.208 standard.

object management (OM). The creation, examination, change, and deletion of potentially complex information objects.

object name. CDS: A name for a network resource.

object UUID. RPC: The Universal Unique Identifier (UUID) that identifies a particular RPC object. A server specifies a distinct object UUID for each of its RPC objects. To access a particular RPC object, a client uses the object UUID to find the server that offers the object. See *object*.

OID. Object identifier.

opaque. A datum or data type whose contents are not visible to the application routines that use it.

Open Software Foundation (OSF). A nonprofit research and development organization set up to encourage the development of solutions that allow computers from different vendors to work together in a true open-system computing environment.

Open Transaction Manager Access (OTMA). A transaction-based connectionless client-server protocol whose implementation is specific to IMS/ESA in an MVS sysplex environment. It uses the MVS Cross-System Coupling Facility (XCF) as its transport layer.

operation. (1) GDS: Processing performed within the directory to provide a service, such as a read operation. (2) RPC: The task performed by a routine or procedure that is requested by a remote procedure call.

organization. (1) The third field of a subject identifier. (2) Security: Data that associates a named set of users who can be granted common access rights that are usually associated with administrative policy.

OTMA. Open Transaction Manager Access.

P

PAC. Privilege attribute certificate.

partially bound binding handle. RPC: A server binding handle that contains an incomplete server address lacking an endpoint. Contrast with *fully bound binding handle*.

partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

password. A secret string of characters shared between a computer system and a user. The user must specify the character string to gain access to the system.

PCB. Program communication block.

PDS. Partitioned data set.

permission. (1) The modes of access to a protected object. The number and meaning of permissions with respect to an object are defined by the access control list (ACL) Manager of the object. (2) GDS: One of five groups that assigns modes of access to users: MODIFY PUBLIC, READ STANDARD, MODIFY STANDARD, READ SENSITIVE, or MODIFY SENSITIVE. Synonymous with *access right*. See also *access control list*.

pipe. (1) RPC: A mechanism for passing large amounts of data in a remote procedure call. (2) The data structure that represents this mechanism.

platform. The operating system environment in which a program runs.

position (within a string). XOM: The ordinal position of one element of a string relative to another.

position (within an attribute). XOM: The ordinal position of one value relative to another.

presentation address. An unambiguous name that is used to identify a set of presentation service access points. Loosely, it is the network address of an open systems interconnection (OSI) service.

principal. Security: An entity that can communicate securely with another entity. In the DCE, principals are represented as entries in the Registry database and include users, servers, computers, and authentication surrogates.

privilege attribute. Security: An attribute of a principal that may be associated with a set of permissions. DCE privilege attributes are identity-based and include the principal's name, group memberships, and local cell.

privilege attribute certificate (PAC). Security: Data describing a principal's privilege attributes that has been certified by an authority. In the DCE, the Privilege Service is the certifying authority; it seals the privilege attribute data in a ticket. The authorization protocol, DCE Authorization, determines the permissions granted to principals by comparing the privilege attributes in PACs with entries in an access control list.

privilege ticket. Security: A ticket that contains the same information as a simple ticket, and also includes a privilege attribute certificate. See *service ticket*, *simple ticket*, and *ticket-granting ticket*.

profile. RPC: An entry in a name service database that contains a collection of elements from which name service interface (NSI) search operations construct search paths for the database. Each search path is composed of one or more elements that refer to name service entries corresponding to a given RPC interface and, optionally, to an object. See *NSI profile attribute* and *profile element*.

profile element. RPC: A record in an RPC profile that maps an RPC interface identifier to a profile member (a server entry, group, or profile in a name service database). See *profile*. See also *group*, *interface identifier* and *server entry*.

program communication block (PCB). IMS/ESA: A control block that describes the source or destination of messages, or the view of an IMS/ESA database.

programming interface. The supported method through which customer programs request software services. The programming interface consists of a set of callable services provided with the product.

protocol. A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication.

protocol sequence. Synonym for *RPC protocol sequence*.

Q

quiescent state. Application Support server: The server state wherein the server can process only management calls. The Application Support server enters this state when the server initialization has been completed or when the server is stopped by an administrative client.

R

RACF. Resource Access Control Facility.

read access. CDS: An access right that grants the ability to view data.

recoverable data. Data whose value persists across system shutdowns and failures. Changes made to recoverable data are permanent regardless of system problems. Logging changes to recoverable data is he most common method of ensuring permanence. Changes to data are recorded in a log that can be replayed to return the data to a valid state.

Recoverable Resource Management Services (RRMS). The z/OS system level syncpoint manager. Three system components provide the function: context services, registration services, and resource recovery services (RRS).

register. (1) RPC: To list an RPC interface with the RPC runtime. (2) To place server-addressing information into the local endpoint map. (3) To insert authorization and authentication information into binding information. See *endpoint map* and *RPC interface*.

remote. Pertaining to a device, file or system that is accessed by your system through a communications line. Contrast with *local*.

remote procedure. RPC: An application procedure located in a separate address space from calling code. See *remote procedure call*.

remote procedure call. RPC: A client request to a service provider located anywhere in the network.

Remote Procedure Call (RPC). A DCE component. It allows requests from a client program to access a procedure located anywhere in the network.

request. A command sent to a server over a connection.

resource. Items such as printers, plotters, data storage, or computer services. Each has a unique identifier associated with it for naming purposes.

Resource Access Control Facility (RACF). An IBM licensed program, that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, and logging the detected unauthorized access to protected resources.

Resource Recovery Services (RRS). The system component that provides the services a resource manager calls to protect resources.

RPC. Remote Procedure Call.

RPC control program (RPCCP). An interactive administrative facility for managing name service entries and endpoint maps for RPC applications.

RPC interface. A logical group of operations, data types, and constant declarations that serves as a network contract for a client to request a procedure in a server. See also *interface definition* and *operation*.

RPC protocol. An RPC-specific communications protocol that supports the semantics of the DCE RPC API and runs over either connectionless or connection-oriented communications protocols.

RPC protocol sequence. A valid combination of communications protocols represented by a character string. Each RPC protocol sequence typically includes three protocols: a network protocol, a transport protocol, and an RPC protocol that works with the network and transport protocols. See *network protocol*, *RPC protocol*, and *transfer protocol*. Synonymous with *protocol sequence*.

RPC runtime. A set of operations that manages communications, provides access to the name service database, and performs other tasks, such as managing servers and accessing security information, for RPC applications. See *RPC runtime library*.

RPC runtime library. A group of routines of the RPC runtime that support the RPC applications on a system. The runtime library provides a public interface to application programmers, the application programming interface (API), and a private interface to stubs, the stub programming interface (SPI). See *RPC runtime*.

RPC thread. A logical thread within which a remote procedure call is executed. See *thread*.

RPCCP. RPC control program

RRMS. Recoverable Resource Management Services.

RRS. Resource Recovery Services.

S

SDSRM. Server distributed syncpoint resource manager.

Security Service. A DCE component that provides trustworthy identification of users, secure communications, and controlled access to resources in a distributed system.

segment. One or more contiguous elements of a string.

server. (1) On a network, the computer that contains programs, data, or provides the facilities that other computers on the network can access. (2) The party that receives remote procedure calls. Contrast with *client*.

server application thread. RPC: A thread running the server application code that initializes the server and listens for incoming calls. See *application thread*, *client application thread*, *local application thread*, and *RPC thread*.

server entry. RPC: A name service entry that stores the binding information associated with the RPC interfaces of a particular RPC server and object Universal Unique Identifiers (UUIDs) for any objects offered by the server. See also *binding information*, *NSI binding attribute*, *NSI object attribute*, *object* and *RPC interface*.

server instance. RPC: A server running in a specific address space. See *server*.

server state. Application Support server: The condition of the Application Support after it has been started. The server state may be any of the following, depending on the actions directed to it by the administrator: initializing, quiescent, starting, operating, or stopping.

server stub. RPC: The surrogate calling code for an RPC interface that is linked with server application code containing one or more sets of remote procedures (managers) that implement the interface. See *client stub*. See also *manager*.

service. In network architecture, the capabilities that the layers closer to the physical media provide to the layers closer to the end user.

service ticket. Security: A ticket for a specified service other than the ticket-granting service. See *privilege ticket*, *simple ticket*, and *ticket-granting ticket*.

session. GDS: A sequence of directory operations requested by a particular user of a particular directory

user agent (DUA) using the same session object management (OM) object.

signal. Threads: To wake only one thread waiting on a condition variable. See *broadcast*.

sign-on. (1) A procedure to be followed at a terminal or workstation to establish a link to a computer. (2) To begin a session at a workstation. (3) Same as log on or log in.

simple name. CDS: One element in a CDS full name. Simple names are separated by slashes in the full name.

simple ticket. Security: A ticket that contains the principal's identity, a session key, a time stamp and other information, sealed using the target's secret key. See *privilege ticket*, *service ticket*, and *ticket-granting ticket*.

single sign-on. In z/OS DCE, single sign-on to DCE allows a z/OS user who has already been authenticated to an MVS external security manager, such as RACF, to be logged in to DCE. DCE does this automatically when a DCE application is started, if the user is not already logged in to DCE.

specific. XOM: The attribute types that can appear in an instance of a given class, but not in an instance of its superclasses.

standard. A model that is established and widely used.

string. An ordered sequence of bits, octets, or characters, accompanied by the string's length.

stub. RPC: A code module specific to an RPC interface that is generated by the Interface Definition Language (IDL) compiler to support remote procedure calls for the interface. RPC stubs are linked with client and server applications and hide the intricacies of remote procedure calls from the application code. See *client stub* and *server stub*.

subject identifier (SID). A string that identifies a user or set of users. Each SID consists of three fields in the form person.group.organization. In an account, each field must have a specific value; in an access control list (ACL) entry, one or more fields may use a wildcard.

sync point. Synchronization point.

synchronization point. An intermediate or end point at which an update to one or more of the transaction's protected resources is logically complete and error-free during the processing of a transaction. A point in time from which IMS/ESA or CICS/ESA or an application program can start over if a failure makes recovery necessary.

syntax. (1) XOM: An object management (OM) syntax is any of the various categories into which the OM specification statically groups values on the basis of their form. These categories are additional to the OM type of the value. (2) A category into which an attribute value is placed on the basis of its form. See *attribute syntax*.

sysplex. A set of MVS systems communicating and cooperating with each other to process customer workloads through certain multisystem hardware components and software services.

system time. The time value maintained and used by the operating system.

T

TCP. Transmission Control Protocol

TCP/IP. Transmission Control Protocol/Internet Protocol

thread. A single sequential flow of control within a process.

thread handle. RPC: A data item that enables threads to share a storage management environment.

ticket. Security: An application-transparent mechanism that transmits the identity of an initiating principal to its target. See *privilege ticket*, *service ticket*, *simple ticket* and *ticket-granting ticket*.

ticket-granting ticket. Security: A ticket to the ticket-granting service. See *privilege ticket*, *service ticket*, and *simple ticket*.

TID. Transaction identifier. A unique name for a transaction. The Encina Tran-C component automatically manages passing, generating, and manipulating TIDs.

time provider (TP). DTS: A process that queries universal time coordinated (UTC) from a hardware device and provides it to the server.

TP. Time provider.

transaction. (1) A unit of processing consisting of one or more application programs initiated by a single request, often from a terminal. For Application Support, this refers to an IMS or a CICS transaction. (2) IMS/ESA: A message destined for an application program. (3) For the Encina recovery server, this refers to an IMS transaction.

transactional RPC. Similar to a standard DCE RPC but carrying additional information that identifies the

Encina transaction on whose behalf it is running. A transactional RPC uses DCE RPCs as its underlying communication mechanism but extends this by providing transactional semantics.

transfer syntax. RPC: A set of encoding rules used for transmitting data over a network and for converting application data to and from different local data representations. See also *Network Data Representation*.

Transmission Control Protocol (TCP). A communications protocol used in Internet and any other network following the U.S. Department of Defense standards for inter-network protocol. TCP provides a reliable host-to-host protocol in packet-switched communication networks and in an interconnected system of such networks. It assumes that the Internet Protocol is the underlying protocol. The protocol that provides a reliable, full-duplex, connection-oriented service for applications.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of nonproprietary communications protocols that support peer-to-peer connectivity functions for both local and wide area networks.

two-phase commit protocol. A set of actions that ensures that an application program makes all the changes of a transaction to a collection of resources or makes no changes. The first phase is the prepare phase. One participant coordinates the transaction. After all the other participants have notified the coordinator that they have completed the prepare phase, the coordinator writes a prepare record. This ensures the transaction will complete, even if the system fails. The second phase is the commit phase. The coordinator notifies each participant to commit, and each writes a log record, releases any held resources, and sends the coordinator an acknowledgment. The coordinator informs all participants of the outcome and writes a commit record.

type. XOM: A category into which attribute values are placed on the basis of their purpose. See *attribute type*.

type UUID. RPC: The Universal Unique Identifier (UUID) that identifies a particular type of object and an associated manager. See also *manager* and *object*.

U

unexport. RPC: To remove binding information from a server entry in a name service database. Contrast with *export*.

Universal Unique Identifier (UUID). RPC: An identifier that is immutable and unique across time and space. A UUID can uniquely identify an entity such as an object or an RPC interface. See *interface UUID*, *object UUID*, and *type UUID*.

URID. Unit of Recovery ID. An IMS identifier of a transaction's RRS identity.

user. A person who requires the services of a computing system.

UUID. Universal unique identifier

V

value. XOM: An arbitrary and complex information item that can be viewed as a characteristic or property of an object. See *attribute value*.

Virtual Telecommunications Access Method (VTAM®). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VTAM. Virtual Telecommunications Access Method.

W

workstation. A device that enables users to transmit information to or receive information from a computer, for example, a display station or printer.

X

XDS. The X/Open Directory Services API.

X/Open Directory Services (XDS). An application program interface that DCE uses to access its directory service components. XDS provides facilities for adding, deleting, and looking up names and their attributes. The XDS library detects the format of the name to be looked up and directs the calls it receives to either GDS or CDS. XDS uses the XOM API to define and manage its information.

XCF. Cross-system coupling facility.

XOM. The X/Open OSI-Abstract-Data Manipulation API.

z/OS. A network computing-ready, integrated operating environment consisting of more than 50 base elements and integrated optional features delivered as a configured, tested system.

Bibliography

This bibliography is a list of publications for z/OS DCE and other products. The complete title, order number, and a brief description is given for each publication.

z/OS DCE Publications

This section lists and provides a brief description of each publication in the z/OS DCE library.

Overview

- *z/OS DCE Introduction*, GC24-5911

  This book introduces z/OS DCE. Whether you are a system manager, technical planner, z/OS system programmer, or application programmer, it will help you understand DCE and evaluate the uses and benefits of including z/OS DCE as part of your information processing environment.

Planning

- *z/OS DCE Planning*, GC24-5913

  This book helps you plan for the organization and installation of z/OS DCE. It discusses the benefits of distributed computing in general and describes how to develop plans for a distributed system in a z/OS environment.

Administration

- *z/OS DCE Configuring and Getting Started*, SC24-5910

  This book helps system and network administrators configure z/OS DCE.

- *z/OS DCE Administration Guide*, SC24-5904

  This book helps system and network administrators understand z/OS DCE and tells how to administer it from the batch, TSO, and shell environments.

- *z/OS DCE Command Reference*, SC24-5909

  This book provides reference information for the commands that system and network administrators use to work with z/OS DCE.

- *z/OS DCE User's Guide*, SC24-5914

  This book describes how to use z/OS DCE to work with your user account, use the directory service, work with namespaces, and change access to objects that you own.

Application Development

- *z/OS DCE Application Development Guide: Introduction and Style*, SC24-5907

  This book assists you in designing, writing, compiling, linking, and running distributed applications in z/OS DCE.

- *z/OS DCE Application Development Guide: Core Components*, SC24-5905

  This book assists programmers in developing applications using application facilities, threads, remote procedure calls, distributed time service, and security service.

- *z/OS DCE Application Development Guide: Directory Services*, SC24-5906

  This book describes the z/OS DCE directory service and assists programmers in developing applications for the cell directory service and the global directory service.

- *z/OS DCE Application Development Reference*, SC24-5908

  This book explains the DCE Application Program Interfaces (APIs) that you can use to write distributed applications on z/OS DCE.

Reference

- *z/OS DCE Messages and Codes*, SC24-5912

  This book provides detailed explanations and recovery actions for the messages, status codes, and exception codes issued by z/OS DCE.

z/OS SecureWay® Security Server Publications

This section lists and provides a brief description of books in the z/OS SecureWay Security Server library that may be needed for z/OS SecureWay Security Server DCE and for RACF® interoperability.

- *z/OS SecureWay Security Server DCE Overview*, GC24-5921

  This book describes the z/OS SecureWay Security Server DCE and provides z/OS SecureWay Security Server DCE information about the z/OS DCE library.

- *z/OS SecureWay Security Server LDAP Client Programming*, SC24-5924

  This book describes the Lightweight Directory Access Protocol (LDAP) client APIs that you can use to write distributed applications on z/OS DCE and gives you information on how to develop LDAP applications.

- *z/OS SecureWay Security Server RACF Security Administrator's Guide*, SA22-7683.

  This book explains RACF concepts and describes how to plan for and implement RACF.

- *z/OS SecureWay Security Server LDAP Server Administration and Use*, SC24-5923

  This book describes how to install, configure, and run the LDAP server. It is intended for administrators who will maintain the server and database.

- *z/OS SecureWay Security Server Firewall Technologies*, SC24-5922

  This book provides the configuration, commands, messages, examples and problem determination for the z/OS Firewall Technologies. It is intended for network or system security administrators who install, administer and use the z/OS Firewall Technologies.

Tool Control Language Publication

- *Tcl and the Tk Toolkit*, John K. Osterhout, (c)1994, Addison—Wesley Publishing Company.

  This non-IBM book on the Tool Control Language is useful for application developers, DCECP script writers, and end users.

IBM C/C++ Language Publication

- *z/OS C/C++ Programming Guide*, SC09-4765

  This book describes how to develop applications in the C/C++ language in z/OS.

z/OS DCE Application Support Publications

This section lists and provides a brief description of each publication in the z/OS DCE Application Support library.

- *z/OS DCE Application Support Configuration and Administration Guide*, SC24-5903

  This book helps system and network administrators understand and administer Application Support.

- *z/OS DCE Application Support Programming Guide*, SC24-5902

  This book provides information on using Application Support to develop applications that can access CICS® and IMS™ transactions.

Encina Publications

- *z/OS Encina Toolkit Executive Guide and Reference*, SC24-5919

  This book discusses writing Encina applications for z/OS.

- *z/OS Encina Transactional RPC Support for IMS*, SC24-5920

  This book is to help software designers and programmers extend their IMS transaction applications to participate in a distributed, transactional client/server application.

Index

T

W

Z

Readers' Comments

z/OS
Encina Transactional RPC
Support for IMS™

Publication No. SC24-5920-00

You may use this form to report errors, to suggest improvements, or to express your opinion on the appearance, organization, or completeness of this book.

Date: _____

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note

Report system problems to your IBM representative or the IBM branch office serving you.
U.S. customers can order publications by calling the IBM Software Manufacturing Solutions at **1-800-879-2755**.

In addition to using this postage-paid form, you may send your comments by:

FAX	1-607-752-2327	Internet	pubrcf@vnet.ibm.com
IBMLink	GDLVME(PUBRCF)		

Would you like a reply? ___YES ___ NO If yes, please tell us the type of response you prefer.

___ Electronic address: _____

___ FAX number: _____

___ Mail: (Please fill in your name and address below.)

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments
SC24-5920-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department G60
International Business Machines Corporation
Information Development
1701 North Street
ENDICOTT NY 13760-5553

Fold and Tape | **Please do not staple** | Fold and Tape

SC24-5920-00

Cut or Fold
Along Line

Program Number: 5694-A01

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

SC24-5920-00

Spine information:

z/OS Encina Transactional RPC Support for IMS