

Ingres[®] 2006

Distributed Transaction Processing User Guide

INGRES[®]

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Ingres Corporation ("Ingres") at any time.

This Documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of Ingres. This Documentation is proprietary information of Ingres and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this Documentation for their own internal use, provided that all Ingres copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. The user consents to Ingres obtaining injunctive relief precluding any unauthorized use of the Documentation. Should the license terminate for any reason, it shall be the user's responsibility to return to Ingres the reproduced copies or to certify to Ingres that same have been destroyed.

To the extent permitted by applicable law, INGRES PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL INGRES BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF INGRES IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The use of any product referenced in this Documentation and this Documentation is governed by the end user's applicable license agreement.

The manufacturer of this Documentation is Ingres Corporation.

For government users, the Documentation is delivered with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013 or applicable successor provisions.

Copyright © 2005-2006 Ingres Corporation. All Rights Reserved.

Ingres, OpenROAD, and EDBC are registered trademarks of Ingres Corporation. All other trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

Chapter 1: Introduction

Audience	1-1
Special Considerations	1-1
General Restrictions	1-2

Chapter 2: Overview

Product Description	2-1
The X/Open DTP Standard	2-1
Transaction Processing Products	2-2
The X/Open DTP Model	2-3
XA Application Model	2-3
Ingres Distributed Option	2-5
Installation Requirements	2-5

Chapter 3: Programming Ingres DTP Applications

Binding to Database Servers	3-1
Programming Requirements	3-2
The Include Files	3-3
The xa_switch_t Structure	3-3
SQL Statement Restrictions	3-4
Transaction Context	3-5
Multiple Resource Manager Instances	3-6
Two Phase Commit	3-7
Error Handling	3-7
Error Messages	3-7
Database Access in Error Handlers	3-7
Error Codes	3-7
Application Server Design	3-8

Chapter 4: Troubleshooting and Tuning

Obtaining Trace and Error Information	4-1
Aborting Transactions	4-4

Performance Tuning	4-4
Setting Session Cache Limit	4-4
Performance-Related Settings	4-5

Appendix A: Building CICS/6000 Programs on UNIX

CICS/6000 and Ingres DTP	A-1
Configuring the System	A-2
Step 1: Update the CICS/6000 Region Environments File	A-2
Step 2: Add the Ingres CICS User.....	A-3
Step 3: Register Databases with a CICS/6000 Region	A-3
Step 4: Build the Switch Load File.....	A-4
Step 5: Compile and Link the Switch Load File.....	A-4
Adding COBOL Support.....	A-5
Step 1: Create an Exports List for the Ingres Library	A-5
Step 2: Modify the Link Script	A-6
Step 3: Rebuild the CICS/6000 COBOL Run Time System.....	A-6
Building CICS/6000 Applications	A-7
C Applications.....	A-7
COBOL Applications	A-8
Using Multiple Resource Manager Instances with CICS/6000	A-8

Appendix B: Building Encina Programs on UNIX

Building Encina Programs.....	B-1
Step 1: Preparing the DCE Environment	B-2
Step 2: Registering the Resource Manager Instances.....	B-3
Step 3: Creating the Encina Code	B-3
Step 4: Compiling and Linking the Program.....	B-4
Step 5: Enabling Tracing (Optional)	B-5
Step 6: Running the Program.....	B-5
Step 7: Verifying the Results (Optional).....	B-5
Using TRAN-C	B-5

Appendix C: Building Tuxedo Programs on UNIX

Process Architecture.....	C-1
Installation Requirements	C-2
Configuring the Tuxedo System	C-2
Step 1: Modifying the Resource Manager Definition File	C-3
Step 2: Building the TMS Server.....	C-3

Creating a Tuxedo Application.....	C-3
Step 1: Building Application Servers.....	C-4
Step 2: Editing the Application Configuration File	C-5
Step 3: Editing the ENVFILE.....	C-6
Starting and Shutting Down Application Servers	C-7
Verifying Server Startup	C-7
Application Development Guidelines.....	C-8
Placement of Transaction Demarcation Calls	C-8
Handling Errors	C-8
Handling Deadlock	C-8
TP_COMMIT_CONTROL.....	C-9

Index

Chapter 1: Introduction

Ingres® DTP (distributed transaction processing) is a set of libraries and programming extensions that enable you to create X/Open DTP-compliant applications that access Ingres database servers. The X/Open DTP standard defines how transaction processing is performed in a distributed, open environment.

The *Distributed Transaction Processing User Guide*:

- Describes the features that Ingres DTP provides for developing X/Open DTP-compliant applications.
- Provides additional information to help you use Ingres DTP more effectively.

Audience

This guide is designed for programmers who want to use Ingres DTP to develop database applications that comply with the X/Open DTP standard, and for database administrators who must administer Ingres DTP installations. To use Ingres DTP, you should:

- Understand Ingres products and embedded SQL programming
- Understand the X/Open DTP transaction processing standard
- Be familiar with your particular transaction processing product

Special Considerations

For details about the X/Open DTP model and XA standard, refer to the related X/Open CAE specification, available from the X/Open Company, Ltd:

- Distributed Transaction Processing: The XA Specification
- ISBN: 1 872630 24 3
- X/Open document number XO/CAE/91/300

Your primary resource for creating X/Open DTP-compliant applications is the documentation provided by your Transaction Processing system vendor.

General Restrictions

The following XA options are not supported in this release of Ingres DTP:

- Multithreaded database client libraries
- Dynamic registration
- Asynchronous XA operations

Chapter 2: Overview

This chapter describes the Ingres DTP product and the components of an Ingres DTP application.

Product Description

Ingres DTP enables you to create applications that interact with Ingres database servers in an X/Open DTP environment. Your Ingres DTP application looks much like a standard Ingres embedded SQL application.

The Ingres DTP product consists of header files and a library of routines that you link with your application, plus programming restrictions (described in this guide) that you must observe when you create your application code.

The X/Open DTP Standard

The X/Open DTP standard defines how transaction processing is performed in a distributed, open environment. In this environment, three logical components interact to execute global transactions (logical transactions that may span multiple hardware and software platforms):

- Resource manager

The resource manager (RM) manages access to data (and possibly other shared resources). In an Ingres DTP installation, the resource manager is an Ingres database server acting in combination with Ingres DTP library routines linked into the application.

- Transaction manager

The transaction manager (TM) oversees the execution of global transactions. The transaction manager performs the following functions:

- It accepts global transaction start, commit, and rollback calls from the application program. (Transaction rollback can also be initiated by the transaction manager itself or by the resource manager.)
- It directs resource managers to start, end, prepare, commit, and rollback global transactions. To communicate with resource managers, the transaction manager calls XA routines provided by the resource managers.

- Application program

The application program performs the following functions:

- It interacts with the end-user.
- It notifies the transaction manager when it wants to begin, commit, or abort a transaction. To communicate with the TM, the application program calls routines supplied by the TP vendor.
- It performs database access. To interact with an Ingres database server, the application program uses Ingres embedded SQL.

Transaction Processing Products

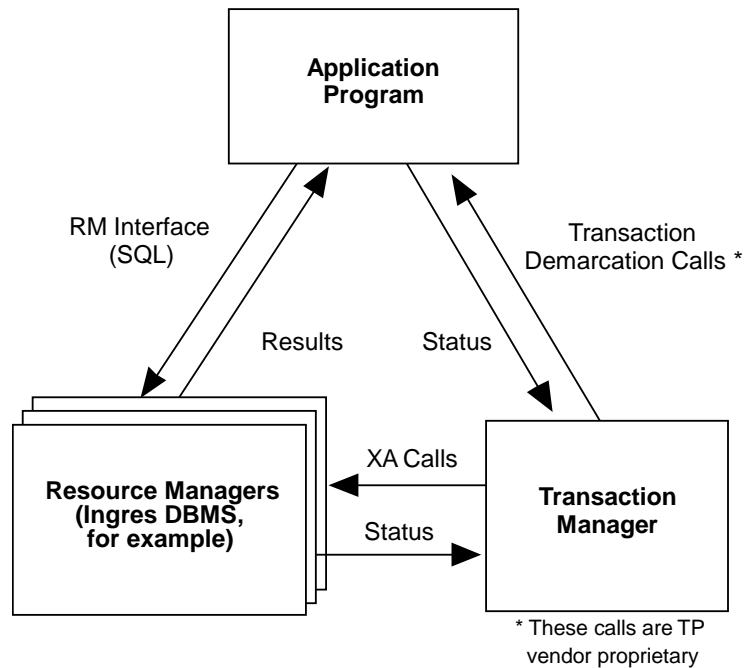
Transaction processing (TP) products provide the following benefits:

- They enable developers to create applications that perform transaction processing across multiple hardware platforms, operating systems, database management systems, and TP monitor environments.
- They can maintain high availability by performing a variety of administrative tasks transparently, such as replicating data, rerouting transactions, and restarting servers that have failed.
- They can be scaled to coordinate transaction processing for a large number of users.
- They can be tuned and monitored for performance.

Many software vendors have introduced X/Open DTP-compliant transaction processing products. For details about using Ingres DTP with a specific TP product, refer to the appendixes at the end of this guide.

The X/Open DTP Model

The following diagram illustrates how the logical components of the X/Open DTP model interact:



XA Application Model

Application programs are typically divided into *application clients* and *application servers*:

- **Application client**

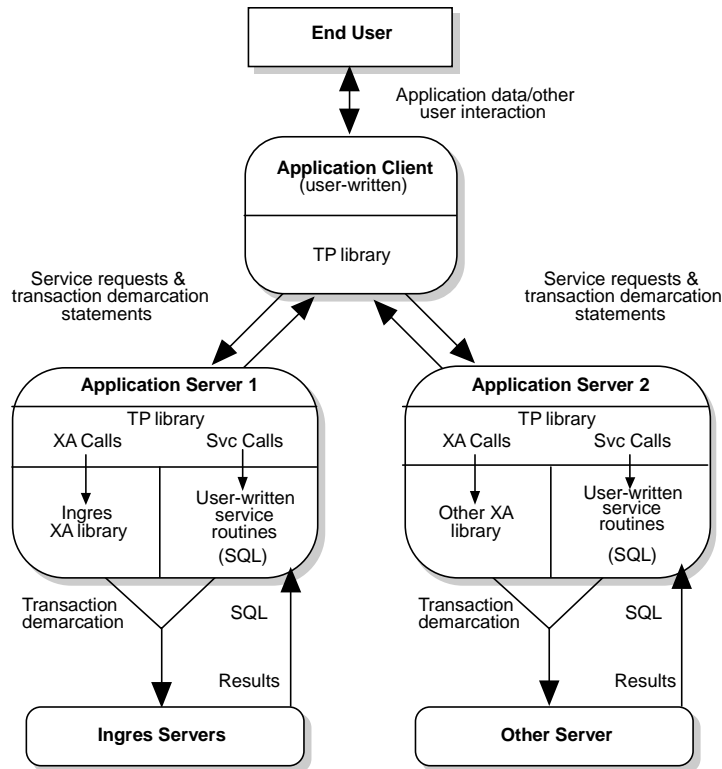
Application clients interact with the end user and request services from application servers through the transaction processing system's transaction manager. The application client demarcates transactions using routines supplied by the transaction processing software vendor. The application client should not interact directly with underlying database management systems.

- **Application server**

Application servers perform services on behalf of remote application clients. Typically these services are registered with the transaction manager. The transaction manager relays transaction demarcation events to the resource manager by calling XA routines.

Both application clients and application servers must be linked with libraries provided by the transaction processing software vendor. Application servers must also be linked with the Ingres DTP XA library routines. Applications programmers need not be concerned with the XA routines, because these calls are performed by the TP system on behalf of the application program. For details about the calls that your application must issue to demarcate transactions, refer to the programmer documentation supplied by your transaction processing software vendor.

The following figure illustrates the structure of a typical Ingres DTP application:



This figure shows an application client that accepts data and other commands from an end user. The application client requests service from two application servers. One of the application servers communicates with several Ingres DBMS servers, and the other with a single, non-Ingres DBMS server. The application client is linked with a library of routines provided by the TP vendor. The application servers are linked with libraries provided by the TP vendor and libraries provided by Ingres Corporation or the non-Ingres DBMS vendor.

Ingres Distributed Option

Ingres® Distributed Option enables you to operate on a combination of local and remote Ingres databases operating in a variety of hardware and software environments. The Distributed Option performs two phase commit for distributed Ingres transactions. Ingres DTP applications can participate in two phase commit transactions coordinated by TP monitors in environments that include both Ingres and other database management systems. Ingres DTP applications and Distributed Option applications can operate simultaneously against the same Ingres DBMS server. The following table compares the Distributed Option with Open TP systems:

Feature	Ingres Distributed Option	Open TP System
Typical application	Distributed decision support applications	Distributed online transaction processing (OLTP) applications
Scope and nature of typical application	Enterprise-wide	Mission-critical
Transaction commitment protocol	Two phase commit across multiple Ingres databases	Heterogeneous two phase commit coordinated by external transaction managers
Application programming interface	SQL	SQL plus TP system application programmer interface (API)

Installation Requirements

The following table lists the software versions and requirements for Ingres DTP:

Feature	Requirement
Ingres database	Version 2.0 or later
Ingres DTP libraries	Version 2.0 or later
Ingres Net	Version 2.0 or later
ESQL preprocessor	Version 2.0 or later
Database requirements	Ingres database with system catalogs upgraded to Version 2.0 or later

Chapter 3: Programming Ingres DTP Applications

This chapter describes Ingres DTP features that you can use to create X/Open DTP-compliant programs that access Ingres database servers. The material in this chapter is presented with the assumption that you are familiar with your XA transaction management programming environment.

You create Ingres DTP applications using SQL statements embedded in C or COBOL programs (depending on what your TP vendor supports). For details about creating embedded SQL programs, see the *Embedded SQL Companion Guide*. For details about configuring and operating specific TP products with Ingres DTP, see the appendixes of this guide.

Binding to Database Servers

At run time, application servers must register with the transaction manager for each resource manager they intend to access. The transaction processing software vendor provides a proprietary mechanism for this purpose. The binding of an application server to a resource manager is referred to as a *resource manager instance* (RMI).

The application server registration routine requires an argument called the *open string*. The Ingres open string has the following format:

```
INGRES [vnode::]databasename [as connection_name]
      [options = flag {, flag}]
```

The open string can contain a maximum of 256 characters and must be null-terminated (C string). (For CICS/6000, the open string is specified as resource information in a stanza file, and therefore cannot be null-terminated.) The following table describes each parameter in the open string:

Parameter	Description	Required or Optional
<i>vnode</i>	The name of the vnode on which the Ingres database resides. (A "vnode" is the Ingres term for "virtual node," a location for a database. For information about vnodes, see the <i>Connectivity Guide</i> .)	Optional
<i>databasename</i>	The name of the Ingres database.	Required

Parameter	Description	Required or Optional
<i>connection_name</i>	A unique string that is associated with a specific binding of a TM with an RMI. If your TP vendor supports multiple RMIs, the connection name is used by your application to switch between RMIs. If no connection name is specified, the default connection name is <code>[vnode:]database</code> .	Optional
<i>flag</i>	Connect-time flags for the Ingres DBMS. For information about these flags, see the description of the <code>sql</code> command in the <i>Command Reference Guide</i> .	Optional

Examples of valid open strings are:

```
INGRES pxa12zbf as personnel
INGRES usa::sec23xyzy as security options = '-uaalex'
```

In the Ingres DTP environment, an application server can bind to more than one resource manager instance but can have only one binding to each resource manager instance. For example, the following bindings would be illegal for a specific application server:

```
INGRES ny::pxa12zbf
INGRES usa::pxa12zbf
```

Assuming that the “ny” and “usa” vnodes represent the same installation, these open strings erroneously bind to the same database.

Programming Requirements

To create an Ingres DTP application, you must:

1. Create or modify the source code according to conventions described in your TP vendor documentation and in this guide.
2. Preprocess the source code using the ESQL preprocessor. For details see the *Embedded SQL Companion Guide*.
3. Compile the resulting source code. For details, see the *Embedded SQL Companion Guide*.
4. Link the resulting object module with:
 - The TP libraries,

- The Ingres DTP libraries (which provide XA entry points for the TM), and
- The Ingres embedded SQL libraries—see the *Embedded SQL Companion Guide* for details.

Note: Application servers must include an SQLCA. For details, see the *SQL Reference Guide*. To correctly build an application server, you must have read access to the `$II_SYSTEM/ingres/files` subdirectory.

The Include Files

All Ingres DTP application servers must include the “`xa.h`” and “`iixa.h`” header files, located in the `$II_SYSTEM/ingres/files` subdirectory. To include the required header files, use the following lines of C code:

```
#include <xa.h>
#include <iixa.h>
```

When you compile your application, you must specify the following option on the `cc` command line:

```
-i $II_SYSTEM/ingres/files
```

The `xa_switch_t` Structure

The “`xa_switch_t`” structure, described in the “`xa.h`” file, contains the set of entry points that enables the transaction manager to call the standard XA routines supplied by the resource manager. You use this structure to register XA entry points with the transaction manager; your TP software vendor provides the registration routine.

The “`xa_switch_t`” structure is declared in the “`iixa.h`” include file. For details about this structure, refer to the X/Open XA documentation.

Note: If you have already included a TP vendor-supplied “`xa.h`” from a different location, omit the Ingres version.

SQL Statement Restrictions

Ingres DTP applications must not issue the following statements in any context: embedded SQL, dynamic SQL, or database procedures (including procedures fired by rules).

Statement	Description
call <i>subsystem</i>	Restricted because a call to an Ingres tool might violate transaction semantics. (However, if your TP software permits forking of processes, your application can issue the call system statement. Call system performs a temporary exit to the operating system.)
commit	Restricted because the transaction manager handles transaction commits.
connect and disconnect	Restricted because connections to databases are replaced by Transaction Processing software vendor-supplied server initialization and shutdown routines.
prepare to commit	Restricted because the transaction manager coordinates two phase commit.
rollback	Restricted because the transaction manager handles transaction commits. (However, your application can issue the rollback to <i>savepoint</i> statement.)
set autocommit	Restricted because the transaction manager handles transaction commits.
set_sql (session)	Use set connection instead.

In Ingres DTP applications, the preceding statements cause run-time errors. If the Ingres DTP application executes a database procedure that contains one of these restricted statements, the offending global transaction is rolled back.

Note: The obsolete Ingres SQL statements begin transaction, end transaction, and abort, are also restricted. Do not use these statements in your application.

Transaction Context

Your application must not issue the following statements in an Ingres multistatement transaction:

- set session authorization
- enable/disable security_audit
- set lockmode
- set [no]logging
- set session with on_error

However, when your application issues these statements, they must be issued with an XA transaction, demarcated as required by your TP vendor software. To determine whether your application is in a transaction, issue the Ingres SQL `inquire_sql(transaction)` statement. The following pseudocode illustrates this method.

```
begin_xa_transaction();
exec sql inquire_sql (:trxflag = transaction);
if trxflag = 0
{
    exec sql set lockmode...
    exec sql set session with on_error...
}
...
```

For details about `inquire_sql`, see the *SQL Reference Guide*.

To avoid conflicts with Ingres transaction states, you can use environment variables to specify a set of SQL statements to be sent to the Ingres DBMS whenever Ingres DTP creates a connection to an Ingres database. The following environment variables are listed in the order in which their corresponding statements are sent to the DBMS:

ING_SYSTEM_SET	Executed for all connections
ING_SET_DBNAME	Executed for connections to the specified database
ING_SET	Executed for all connections

(For details about these environment variables, see the *System Administrator Guide*.) To set the environment variables for your entire Ingres installation, use the `ingsetenv` command. To set the environment variables for your current session, use the `setenv` command. Any set statements issued by your application after a connection is made override the settings established by the preceding environment variables.

Non-DTP Ingres SQL applications establish connections by issuing the connect statement. In Ingres DTP applications, the connect statement is not required (and in fact is not valid). Connections to databases are made by Ingres DTP on behalf of your application—typically (though not always), a connection is established the first time your application access the RMI for which the connection is required.

Multiple Resource Manager Instances

An Ingres DTP application server can support multiple resource manager instances (connections to different Ingres databases). Each connection (binding) must be to a different Ingres database. Ingres DTP applications must not issue the Ingres SQL connect statement. Database connections are defined by registering databases using the open string as required by the Transaction Processing software. For details about the open string, see [Binding to Database Servers](#) in this chapter.

To interact with different databases, your application must issue the set connection statement, described in the *SQL Reference Guide*. For applications that use multiple sessions, service routines must issue the set connection statement at the beginning of the routine (or when resuming a suspended association) to ensure that the routine is interacting with the correct resource manager instance.

The following code illustrates the use of multiple resource manager instances in a sample application. In this example, money is transferred from one account to another.

```
exec sql set connection 'conn1';
exec sql update account
  set balance = balance + :transfer_amount
  where acct = :acct;
exec sql set connection 'conn2';
exec sql update account
  set balance = balance - :transfer_amount
  where acct = :acct;
```

Note: To determine whether your TP product supports multiple resource manager instances, refer to your TP vendor documentation. Note that, even if your application does not interact with multiple RMIs, you must issue the set connection statement before attempting to access an RM.

Two Phase Commit

Ingres DTP applications must not attempt to manage two phase commit. The Transaction Manager assumes responsibility for global transactions and performs two phase commit on behalf of the application, in a manner that is not visible to the application.

Error Handling

The following sections provide information about handling errors in Ingres DTP applications.

Error Messages

By default, Ingres displays error messages on the terminal. To suppress display of error messages you can define an error handling routine. Your application program must issue the `set_sql(errorhandler)` statement to enable error trapping.

Database Access in Error Handlers

If your error handler performs database access (for example, logs errors to a table), the error handler must explicitly set the connection upon entry and restore the original connection before exiting. For details about handling errors in Ingres SQL, see the *SQL Reference Guide*.

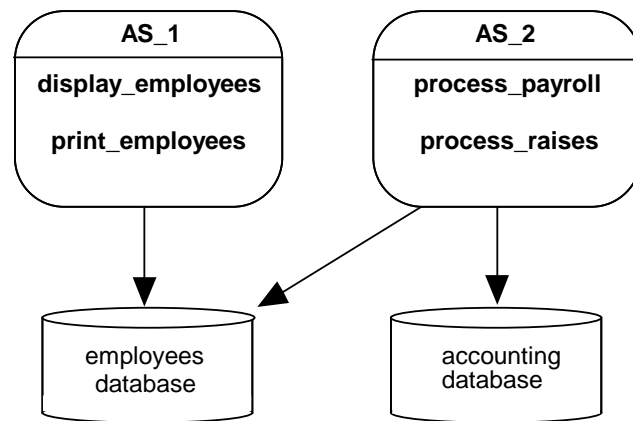
Note: To avoid the overhead of database access in a TP environment, consider logging errors to flat files.

Error Codes

The values of the status variables `SQLSTATE` and `SQLCODE` are set after the execution of an SQL statement. Until an SQL statement is executed, you cannot assume that these values are significant. In particular, you should not assume these values are significant upon entry to a service routine or after suspending and then resuming an XA transaction.

Application Server Design

To minimize the overhead required to make connections on behalf of global transactions, it is recommended that you group transactions that require access to the same databases into application servers. For example, if the “display_employees” and “print_employees” services require access to the “employees” database, and the “process_payroll” and “process_raises” services require access to both the “employees” database and the “accounting” database, design your application servers as shown in the following diagram:



Chapter 4: Troubleshooting and Tuning

This chapter describes tools and techniques for troubleshooting and performance tuning of Ingres DTP applications.

Obtaining Trace and Error Information

To obtain information about XA transaction execution and errors, you can use the following features:

- Ingres error logs

Error logs list errors that have occurred for an Ingres facility. For details about enabling error logs, see the *Database Administrator Guide*.

- Ingres trace files

Trace files contain messages issued by an Ingres facility, enabling you to examine a history of the execution. The GC log records connections to the DBMS, and is therefore of interest to the XA System Administrator. To enable GC tracing, use the II_GCx_TRACE and II_GCA_LOG environment variables. For details, see the *System Administrator Guide*.

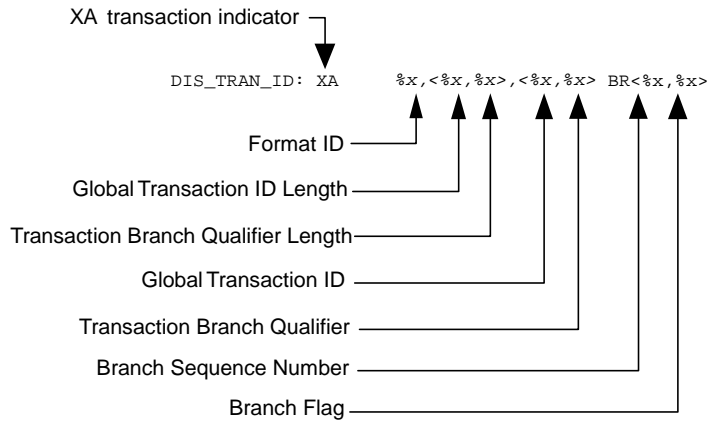
- II_EMBED_SET environment variable

This environment variable allows you to enable a variety of tuning and troubleshooting features. For details, see the *System Administrator Guide*.

- Logstat utility

The logstat utility, described in detail in the *System Administrator Guide*, displays status information about the logging system. Logstat indicates XA transactions by preceding the transaction ID with "XA."

The following figure explains the format of logstat output for XA global transactions. In this diagram, “%x” denotes a 4-byte hexadecimal number.



■ XA trace file

The XA trace file contains XA-related messages from the XA routines, including queries issued by the application server and error messages issued by the DBMS. To enable this feature, set the II_XA_TRACE_FILE environment variable to a valid file name.

To include the process ID of the application server process in the trace file name, specify “%p” as part of the name. For example:

```
setenv II_XA_TRACE_FILE svr_grp_1_%p.trc
```

will produce trace files with names like “svr_grp_1_23145.trc,” “svr_grp_1_23172.trc,” and “svr_grp_1_23229.trc,” where “23145,” “23172,” and “23229” are the process IDs. If you want to use the “%p” feature in a UNIX environment, be sure that the file name you specify doesn’t contain any other percent signs (%).

You must set II_XA_TRACE_FILE before the application server is started. When XA trace logging is enabled, logs XA calls from the transaction manager to the application server and SQL statements issued by the application. For details about setting environment variables, see the *System Administrator Guide*.

The following listing shows a sample portion of XA trace output:

```

---- XA trace: Started at Mon Aug  9 12:08:48 1994 ---

XA_OPEN: flags = TMNOFLAGS
rmiid: 5
OPEN string is: INGRES ticketing as ticketing options= -umoeh
return value: XA_OK
-----
XA_OPEN: flags = TMNOFLAGS
rmiid: 10
OPEN string is: INGRES personnel options = -umoeh
return value: XA_OK
-----
XA_START: flags = TMNOFLAGS
XID: 00000001:4:1:4C4D6C6D:30000000:XA
rmiid: 5
connected to ticketing ( Ingres DTP session 1)
return value: XA_OK
-----
XA_START: flags = TMNOFLAGS
XID: 00000001:4:1:4C4D6C6D:30000000:XA
rmiid: 10
connected to personnel ( Ingres DTP session 2)
return value: XA_OK
-----
Appl Code:
  set connection to rmi: personnel (Ingres DTP session 2)
-----
Appl Code:
  drop table employees
-----
XA_CLOSE: flags = TMNOFLAGS
rmiid: 10
CLOSE string is: INGRES personnel options = -umoeh
Last XA_CLOSE, shutting down XA and freeing session cache
disconnecting personnel (Ingres DTP session 4)
'release session' sent to RM
disconnecting ticketing (Ingres DTP session 3)
'release session' sent to RM
disconnecting personnel (Ingres DTP session 2)
'release session' sent to RM
disconnecting ticketing (Ingres DTP session 1)
'release session' sent to RM
closing XA trace file...
-----

```

If you are using XA trace files with multiple application servers, you can prevent them from overwriting each other's trace files by starting the servers from different directories or by resetting the `II_XA_TRACE_FILE` logical before starting each server.

Aborting Transactions

You can manually abort transactions using the `lartool` utility, described in detail in the *Command Reference Guide*. If you use `lartool` to abort an XA transaction, the transaction manager cannot use its recovery mechanisms to gracefully abort or retry the transactions. If you manually abort a transaction that is part of a global transaction being administered by a transaction manager, you risk database inconsistency.

Performance Tuning

The following section describes a setting that you can alter to maximize the performance of your Ingres DTP applications.

Setting Session Cache Limit

An application server maintains a number of free sessions in order to minimize connection time when an application requires a connection. By default, Ingres DTP accumulates sessions to a maximum of 32. To change the maximum number of free sessions, set the `II_XA_SESSION_CACHE_LIMIT` variable. The value you specify for `II_XA_SESSION_CACHE_LIMIT` cannot exceed the session limit.

To calculate the maximum number of connections your application server will require, use the following formula:

(# of RMI's per AS) * (# of concurrent transactions)

Performance-Related Settings

To modify configuration settings, use Configuration-By-Forms. For Ingres DTP applications, the following settings are important:

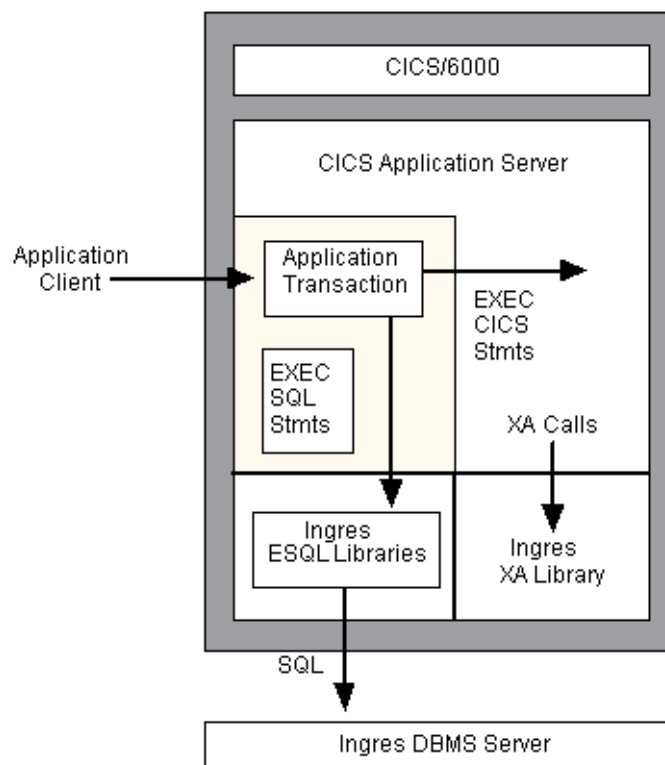
CBF Screen	CBF Setting	Description
Name Server	max_sessions	Maximum number of connections the Name Server can accept. Connections to the Name Server are made when applications start up, so configure this value with the maximum number of simultaneously starting applications in mind.
DBMS Server	connect_limit	Maximum number of connections a DBMS Server can accept.
Communications Server	ib_max	Maximum number of inbound connections the Communications Server can accept.
	ob_max	Maximum number of outbound connections the Communications Server can create.

Appendix A: Building CICS/6000 Programs on UNIX

This appendix tells you how to build Ingres DTP applications for CICS/6000. Where necessary, these instructions refer to related CICS/6000 documentation. The Configuring the System and Adding COBOL Support sections in this appendix are of particular relevance to system administrators. The Building CICS/6000 Applications section in this appendix contains information required by application programmers.

CICS/6000 and Ingres DTP

The following figure shows how Ingres DTP functions in a CICS/6000 environment:



Configuring the System

The following section tells you how to configure CICS/6000 with Ingres DTP. Before performing these configuration procedures, you must have installed CICS/6000, the Ingres DBMS Server, and the Ingres DTP software.

To configure CICS/6000 with Ingres DTP, perform the following steps:

1. Update the CICS/6000 region environments file.
2. Add the CICS user to Ingres.
3. Register databases.
4. Build the Ingres DTP SwitchLoadFile.
5. Compile and link the Ingres DTP SwitchLoadFile.
6. Add COBOL support (optional).

These steps are described in detail in the following sections.

Step 1: Update the CICS/6000 Region Environments File

Environment variables used by Ingres tools (such as Query-By-Forms) or Ingres applications must be declared in the following CICS/6000 file:

```
/var/cics_regions/region_name/environment
```

The value *region_name* must be replaced with the name of the CICS region being configured. In this file, the entries must be in the form *variable=value*. Variable substitution in this file is not permitted; the entries must be specified in full.

The following sample environment file illustrates the entries required for an Ingres installation located in the `"/install/65"` directory.

```
# INGRES Definitions
#
II_SYSTEM=/install/65
PATH=/install/65/ingres/bin:other_path_entries...
LIBPATH=/install/65/ingres/lib:other_library_entries...
```

After updating the CICS region's environment file, the definition of transactions and associated programs to the CICS/6000 region can be performed as described in the *CICS/6000 Customization and Operation Guide*.

Step 2: Add the Ingres CICS User

By default, CICS/6000 applications access Ingres databases using the effective Ingres user name "cics" (unless the 'options =-username' flag is specified in the XA open string). To add the "cics" user to the Ingres installation, perform the following steps:

1. At the operating system prompt, invoke the SQL Terminal Monitor and connect to the iidbdb:

```
isql iidbdb
```

2. Issue the following create user statement:

```
create user cics
  with privileges = (createdb,security,operator);
```

For more information on the create user statement, see the *SQL Reference Guide*.

Step 3: Register Databases with a CICS/6000 Region

For each database that an application intends to access, you must add resource information to the CICS/6000 stanza file for the region in which the application will operate. You can add stanza files using either SMIT panels or the cicsadd command. (For details about SMIT panels, refer to the CICS/6000 documentation.)

The following example adds an XAD resource definition called "ingres" to a CICS/6000 region named "demo1", using the cicsadd command. (For a full description of the XAD attributes, refer to the *CICS/6000 Customization & Operation Guide*.)

```
cicsadd -c xad -r DEMO1 -P INGRES \
  ActivateOnStartup=yes
  ResourceDescription="INGRES XA Sample Definition" \
  SwitchLoadFile=iixa
  XAOpen="ingres demo1db as 'conn1'"
  XAClose="" \
  XASerialize=all_operations;
```

The preceding example specifies a switch load file named "iixa" that resides in the cics default directory, and an open string that maps the "demodb1" database to the "conn1" connection name. For details about the open string, see [Binding to Database Servers](#) in the chapter "Programming Ingres DTP Applications." For details about the switch load file, see [Step 4: Build the Switch Load File](#) in this appendix.

Step 4: Build the Switch Load File

The SwitchLoadFile attribute (illustrated in the cicsadd command shown in the preceding step) must specify an object file that contains the “xa_switch_t” structure definition. This structure provides a table of XA entry points for use by the transaction monitor (TM) libraries. The Ingres DTP switch load file must be defined to:

- Set “cics_xa_switch” to point to the XA switch structure
- Call the CICS XA function cics_xa_init

You can use the following C code to build an Ingres DTP switch load file. To correspond with the cicsadd example in the preceding step, store this code in a text file named “iixa.c”.

```
/* iixa.c */
#include <stdio.h>
#include <xa.h> /* Encina XA header file */
#include <iixa.h> /* INGRES XA header file */
extern struct xa_switch_t RegXA_xa_switch;
extern struct xa_switch_t *cics_xa_switch;
struct IIXA_SWITCH *iixa(void)
{
    cics_xa_switch = &iixa_switch;
    cics_xa_init();
    return(&RegXA_xa_switch);
}
```

Step 5: Compile and Link the Switch Load File

The Ingres DTP switch load file must be compiled and linked into the module name and location that is specified in the CICS/6000 XAD, the XAD stanza file that holds the XA definitions for the CICS/6000 region. To compile and link “iixa.c” into the “iixa” file located in the current directory, issue the following command:

```
xlc_r -v -o iixa -e iixa \
-I/usr/lpp/encina/include/tmxa \
-L/usr/lpp/cics/v2.0/lib \ /*CICS library version specific*/
-L$IISYSTEM/ingres/lib \
-I$IISYSTEM/ingres/files \
-lregxart -lm \
-lq.1 \ /*INGRES XA library*/
iixa.c
```

You can determine the CICS/6000 library version from the CICS/6000 documentation. After completing the preceding steps, you can compile and link applications written in C. For details on how to verify that the configuration has been successfully completed, see [C Applications](#) in this appendix.

Adding COBOL Support

To add support for COBOL, you must:

- Perform the steps described in the Configuring the System section in this appendix.
- Install and configure COBOL as described in the CICS/6000 and COBOL documentation.

To add COBOL support, perform the following steps:

1. Create an exports list for the Ingres shared libraries.
2. Modify the CICS/6000 link script.
3. Rebuild the CICS/6000 COBOL Run Time System (RTS).

The following sections describe these steps in detail.

Step 1: Create an Exports List for the Ingres Library

You must create an exports list that contains a list of Ingres symbols that enables the binding of the Ingres shared libraries into the COBOL Run Time System. To create the exports list perform the following steps:

1. Log in as the installation owner.
2. At the operating system prompt, issue the following commands:

```
echo "#!libcompat.1.a(libcompat.1.o)" > libingres.1.exp
/usr/ucb/nm -p $II_SYSTEM/ingres/lib/libcompat.1.a | \
awk ' $2=="D"||$2=="B" { print $3 } ' | \
sort -u >> libingres.1.exp
echo "#!libq.1.a(libq.1.o)" >> libingres.1.exp
/usr/ucb/nm -g $II_SYSTEM/ingres/lib/libq.1.a | \
awk ' $2=="D"||$2=="B" { print $3 } ' | \
sort -u >> libingres.1.exp
echo "#!libframe.1.a(libframe.1.o)" >> libingres.1.exp
/usr/ucb/nm -g $II_SYSTEM/ingres/lib/libframe.1.a | \
awk ' $2=="D"||$2=="B" { print $3 } ' | \
sort -u >> libingres.1.exp
```

The preceding commands extract symbol table information, remove unnecessary entries, filter out the name column, and sort the resulting list. The exports list will now exist in the current directory where the commands were issued. The following steps assume that the exports list resides in the \$II_SYSTEM/ingres/lib directory.

Step 2: Modify the Link Script

You must modify the CICS/6000 “cicsmkcobol” script to include additional linker commands that use the exports list created in the previous step. Perform the following steps:

1. Make a backup of the “cicsmkcobol” script file.
2. Log in under the “root” account.
3. Edit the “cicsmkcobol” file. Change the following “cob” command from this:

```
cob -${FFLAG}x -o $OUTPUTFILE $OBJECTS -Q "$LD_FLAGS"\
$LD_PATH $CICSLIBS $ARGUMENTS $LIBRARIES
```

to this:

```
cob -${FFLAG}x -o $OUTPUTFILE $OBJECTS -Q "$LD_FLAGS"\
$LD_PATH -Q"-bE:$II_SYSTEM/ingres/lib/libingres.1.exp"\
-Q "-bM:SRE" -Q "-T512" -Q "-H512" \
$CICSLIBS $ARGUMENTS $LIBRARIES
```

Note that the value of \$II_SYSTEM is hard-coded into the above command.

Step 3: Rebuild the CICS/6000 COBOL Run Time System

To rebuild the CICS/6000 COBOL runtime system:

1. Log in under the “root” account.
2. Run the “cicsmkcobol” script that you modified in step 2, specifying the names of the Ingres shared libraries as parameters. To do this, issue the following commands:

```
# cd /usr/lpp/cics/v2.0/bin
# export DEBUG=1
# cicsmkcobol $II_SYSTEM/ingres/lib/libq.1.a\
               $II_SYSTEM/ingres/lib/libcompat.1.a\
               $II_SYSTEM/ingres/lib/libframe.1.a
```

The “cicsmkcobol” script creates a file named “cicsprCOBOL” in the /usr/lpp/cics/v2.0/bin directory.

After performing these steps, you can compile and link applications written in COBOL. For details on how to verify that the configuration has been successfully completed, see [COBOL Applications](#) in this appendix.

Building CICS/6000 Applications

The following sections illustrate the process of building Ingres DTP applications that run with CICS/6000.

C Applications

The following example illustrates the addition of an embedded C application to the CICS/6000 environment using a makefile. The application filename is "app1.sc". To execute the makefile, type the following commands:

```
make app1
make install
```

The source of the makefile is shown in the following example:

```
#-----
# DEFINE VARIABLES
#-----
II_SYSTEM=/install/65
CICSREGION=DEM01
INGHDRS=-I$(II_SYSTEM)/ingres/files
INGLIBS=-L$(II_SYSTEM)/ingres/lib -lcompat.1 -lframe.1 -linterp.1 -lq.1
#-----
# The embedded C PROGRAM
#-----
app1
app1: app1.ccs
    CCFLAGS="$(INGLIBS) -lm"; \
    export CCFLAGS; \
    cicstcl -e -d -lC app1.ccs
    rm -f app1.c
app1.ccs: app1.sc
    esqlc app1.sc
    mv app1.c app1.ccs
#-----
# ADD CICS REGION DEFINITIONS
#-----
install:
    cicsadd -c td -r $(CICSREGION) -P APP1 ProgName=APP1 \
        RSLCheck=none
    cicsadd -c pd -r $(CICSREGION) -P APP1 PathName=$(PWD)/app1
```

COBOL Applications

The following example illustrates the use of a makefile to add an embedded COBOL application to the CICS/6000 environment. The application filename is "app2.scb". To run the makefile, type the following commands:

```
make app2
make install
The source of the makefile is shown in the following example:
#-----
# DEFINE VARIABLES
#-----
II_SYSTEM=/install/65
CICSREGION=DEMO1
INGHDRS=-I$(II_SYSTEM)/ingres/files
INGLIBS=-L$(II_SYSTEM)/ingres/lib -lcompat.1 -lframe.1 -linterp.1 -lq.1
#-----
# The COBOL PROGRAM
#-----
app2: app2.ccp
    cicstcl -e -d app2.ccp
    rm -f app2.cbl
app2.ccp app2.scb
    esqlcbl app2.scb
    mv app2.cbl app2.ccp
#-----
# ADD CICS REGION DEFINITIONS
#-----
install:
    cicsadd -c td -r $(CICSREGION) -P APP2 ProgName=APP2 \
        RSLCheck=none
    cicsadd -c pd -r $(CICSREGION) -P APP2 PathName=$(PWD)/app2
```

Using Multiple Resource Manager Instances with CICS/6000

Ingres DTP supports access to multiple resource manager instances (RMIs) from a single CICS/6000 application server (AS). You must configure each RMI as a separate XAD definition. For more information on configuring multiple RMIs, refer to the *CICS/6000 Customization and Operation Guide*.

In an Ingres DTP application that accesses multiple RMIs, use the SQL set connection statement to specify the RMI to which the application requires access.

The example below shows an Ingres DTP application with two RMIs. The RMIs have been assigned the identifiers CONN1 and CONN2. The following application code example updates both RMIs.

Appendix B: Building Encina Programs on UNIX

This appendix explains how to build Ingres DTP applications for Encina.

Building Encina Programs

The following sections explain how to create an Ingres DTP application that runs with Encina. You must be familiar with both Ingres embedded SQL programming and with Encina Monitor programming.

Before performing the steps below, you must have:

- Installed and started Ingres 2.0 or later, and created the required databases, users, and tables
- Installed DCE (distributed computing environment)
- Installed Encina (version 1.03B or later)
- Performed the Encina DCE configuration procedure (described in the Encina documentation)
- Started the Encina monitor

To create and run an application, you must perform the following steps:

5. Prepare the DCE environment.
6. Register the resource manager instances.
7. Create the Encina application program code.
8. Compile and link the application program.
9. Enable XA tracing (optional).
10. Run the program.
11. Verify the results (optional).

The following sections describe these steps in detail.

Step 1: Preparing the DCE Environment

Before performing the steps in this section, ensure that you have configured DCE and Encina, and have the Encina Monitor (cell and node) manager servers running. For more information, refer to the *Encina Monitor System Administrator's Guide and Reference*.

To prepare the DCE environment, perform the following steps:

1. Modify your PATH variable to include your DCE and Encina bin directories, as well as the Ingres "bin" directory, \$II_SYSTEM/ingres/bin.
2. Create an OS account for the user that you want to run the application.
3. Create the corresponding Ingres user using the Terminal Monitor. Issue the following commands:

```
sql iidbdb
* create user username with privileges=(createdb); \g
* \q
```

4. Login to DCE as the cell administrator, and create a special DCE account and group from which to run your Ingres application servers:

```
% dce_login cell_admin cell_admin_password
% rgy_edit
Current site is: ...
rgy_edit=> domain group
Domain changed to: group
rgy_edit=> add dtp_group -f "DTP user's group"
rgy_edit=> domain principal
Domain changed to: principal
```

5. Add the DCE user:

```
rgy_edit=> add username -f "Ingres Applications Principal"
rgy_edit=> domain account
Domain changed to: account
rgy_edit=> add username -g dtp_group -o none \
-mp <cell_admin_pw> -pw ingapp_password \
-m "Ingres Applications Account"
rgy_edit=> exit
```

6. Add access control lists (ACLs) for the new group:

```
acl_edit ../encina -m group:dtp_group:rt
acl_edit ../encina -ic -m group:dtp_group:rt
acl_edit ../encina -io -m group:dtp_group:rt
```

Step 2: Registering the Resource Manager Instances

To register your Ingres databases with Encina, use the Encina monadmin utility. At the operating system prompt, issue the following command for each database to which your application requires access:

```
% monadmin create rm INGRES -open "open_string"
```

For details about the format of the open string, see [Binding to Database Servers](#) in the chapter “Programming Ingres DPT Applications.” You must register each database to which your application requires access.

For example:

```
% monadmin create rm inventory \  
-open "INGRES chicago::inv07a1 as inventory"  
% monadmin create rm billing \  
-open "INGRES london::bil12c2 as billing"
```

Step 3: Creating the Encina Code

To create an Ingres DTP-compatible application program, you must observe the following requirements when coding:

- Your application must include the “iixa.h” file provided by Ingres. For example, in a C program, use the following code:

```
#include <iixa.h>
```
- Your application must include an SQLCA for error trapping; use the following embedded SQL code:

```
exec sql include sqlca;
```
- Your application must use Encina’s mon_InitResourceManager function to initialize each Ingres database to which it requires access. The argument to the mon_InitResourceManager function must be the resource manager name you specified in the monadmin command.
- In applications that access multiple resource managers, use the Ingres SQL set connection statement to specify the resource manager you want to access. The connection name must be the connection name specified in the open string. For example:

```
void check_inventory (...)  
{  
  exec sql set connection 'inventory';  
  Perform database access
```

- Encina requires your application server to provide a `server_Init` function that initializes application servers. For details about `server_Init`, refer to the *Encina Monitor Programmer's Guide*. The following code illustrates the framework of a typical `server_Init` function:

```
# include <xa.h>
# include <iixa.h>
void server_Init(argc, argv)
int  argc;
char *argv[];
{
    ...
    /* register resource managers */
    mon_ServerRecoverable();
    if (mon_InitResourceManager(&iixa_switch, "inventory")) != MON_SUCCESS)
    {
        handle error routine
    }
    if (mon_InitResourceManager(&iixa_switch, "billing"))
        != MON_SUCCESS)
    {
        handle error routine
    }
    ...
}
```

If your application registers for more than one resource manager, your service routines must issue the SQL set connection statement (to establish which RMI they are accessing) before performing any database access.

Step 4: Compiling and Linking the Program

Encina Applications are generally compiled using standard UNIX makefiles. For details, refer to the *Encina Monitor Programmer's Guide*. To build application servers that use Ingres Embedded SQL, you must modify the makefiles as follows:

- Add the following lines to precompile your embedded SQL source into C source.

```
sourcefile.c: sourcefile.sc
$(II_SYSTEM)/ingres/bin/esqlc sourcefile.sc
```

- Add `$(II_SYSTEM)/ingres/files` to the header file list specified for the `cc` command's `-I` flag
- Add `$(II_SYSTEM)/ingres/lib/libingres.a` to the end of the list of Encina libraries. For example, in the Encina TPM demo "Makefile," this list is specified by the variable `SYS_LIBS`.

Step 5: Enabling Tracing (Optional)

The Ingres DTP tracing feature enables you to verify that the database connections required by your application were successful. (This step is optional.)

To enable tracing:

1. Switch to the window from which the Encina monitor was started.
2. Issue the following command:

```
setenv II_XA_TRACE_FILE trace_file
```

where *trace_file* specifies the name of the file to which Ingres DTP will write trace information. For details about XA trace files, see [Obtaining Trace and Error Information](#) in the chapter “Troubleshooting and Tuning.”

Step 6: Running the Program

To start your application server, perform the following steps:

1. Register the application server with Encina for invocation using the monadmin create server command.
2. Register the application server interfaces with Encina using the monadmin create interface command.
3. Invoke an instance of the application server using the monadmin start server command.
4. Verify that the application server has started using the monadmin query server command.

Step 7: Verifying the Results (Optional)

To verify that your application successfully connected to the Ingres databases, examine the contents of the file that you specified as the XA trace file. For details about error logging and tracing for Ingres applications, see [Obtaining Trace and Error Information](#) in the chapter “Troubleshooting and Tuning.”

Using TRAN-C

If you develop Encina applications using TRAN-C instead of the Encina Monitor API, observe the following programming conventions:

- Set scheduling policy to “exclusive.” To do this, call the “mon_SetSchedulingPolicy” routine with “MON_EXCLUSIVE” as the argument.

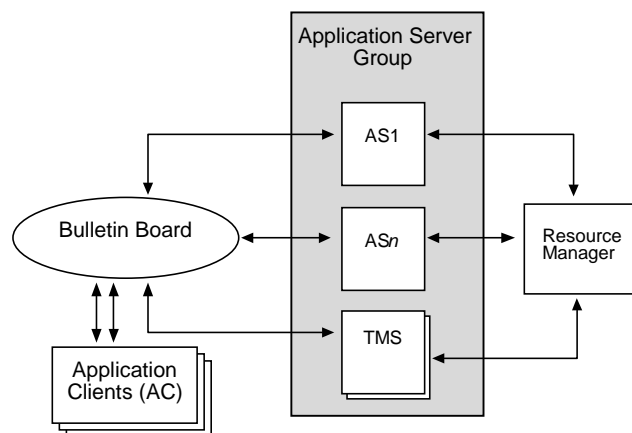
- Do not change the setting for thread support. The default for thread support is "TMXA_SERIALIZE_ALL_XA_OPERATIONS."
- Your application must call "tmxa_RegisterRMI" once (and only once) for each RMI to be accessed by the application. These calls must be issued in the same thread, and must be issued before calling "tmxa_Init." Use the following settings for "tmxa_RegisterRMI" parameters:
 - openInfo: valid open string (see [Binding to Database Servers](#) in the chapter "Programming Ingres DTP Applications")
 - closeInfo: same as openInfo

Appendix C: Building Tuxedo Programs on UNIX

This appendix tells you how to configure the Tuxedo with Ingres DTP, and how to build Ingres DTP applications that interact with Tuxedo. Where necessary, you are referred to related Tuxedo and Ingres documentation. This appendix assumes you are familiar with the Ingres database and Tuxedo.

Process Architecture

The following diagram shows the process architecture of a typical Tuxedo application:



Multiple application client programs communicate with multiple application servers that are combined into *server groups*. Messages are routed between clients and servers through the Tuxedo bulletin board. Transaction manager servers in each server group manage two phase commit protocol and recovery of global transactions.

In this example, the application has been configured into several server groups, and each server group is accessing a different resource manager. Every server in a server group communicates with the resource manager assigned to that group.

Installation Requirements

The following files and directories must be present in your Ingres installation. These files and directories are created when you install Ingres and Ingres DTP.

File Name and Location	Description
<code>\$II_SYSTEM/ingres/utility/iimktms</code>	Ingres TMS build script
<code>\$II_SYSTEM/ingres/lib/libiitux.a</code>	Ingres DTP Tuxedo database client libraries
<code>\$II_SYSTEM/ingres/files/iitxxa.h</code>	X/Open XA switch definition

Note: The first time Tuxedo is used with Ingres, a user table named `tuxedo` will be created in the `iidbdb` database.

Configuring the Tuxedo System

The following section tells you how to configure the Tuxedo system with Ingres DTP. Before performing these configuration procedures, you must have installed Tuxedo, the Ingres database, and the Ingres DTP software. To configure Tuxedo, perform the following steps:

- Modify the Resource Manager Definition file.
- Create an Ingres TMS server executable.

The following sections describe these steps in detail.

Step 1: Modifying the Resource Manager Definition File

The resource manager definition file is called “RM” and is located in the “udataobj” directory of your Tuxedo installation. Add the following two lines to the file:

```
INGRES:iitux_switch:${II_SYSTEM}/ingres/lib/libiitux.a  
${II_SYSTEM}/ingres/lib/libingres.a -lm  
  
INGRES/TMS:iitux_switch:${II_SYSTEM}/ingres/lib/libiitux.a  
${II_SYSTEM}/ingres/lib/libingres.a -lm
```

The lines above are wrapped for editorial reasons. Your modifications must be without breaks. Each line must start with the word INGRES. Each file name must be separated from the next by a space. Failure to follow these instructions will cause server builds to fail, or prevent servers from functioning correctly.

For more details about the resource manager definition file, refer to the Tuxedo documentation.

Step 2: Building the TMS Server

To build a TMS executable for use with an Ingres database, use the build script supplied by Ingres. (After you upgrade Ingres DTP and Tuxedo, you need to rebuild your executables.) The script must be executed from the Tuxedo administrator’s account. To execute the build script, issue the following command:

```
II_SYSTEM/ingres/utility/iimktms
```

The script builds a Tuxedo TMS executable that accesses the Ingres DTP client libraries required to interact with an Ingres database. The executable image is called “TMS_INGRES,” and is located in the “bin” directory of your Tuxedo installation (\$ROOTDIR/bin).

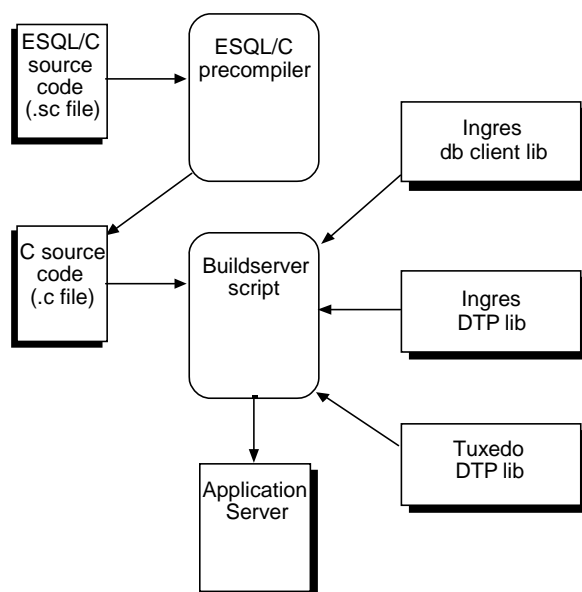
Creating a Tuxedo Application

To create a Tuxedo application that uses an Ingres resource manager, you must perform the following steps in addition to the standard Tuxedo configuration and setup issues:

1. Build application server executables.
2. Edit the application configuration (“ubbconfig”) file.
3. Edit the application ENVFILE.

Step 1: Building Application Servers

For an application server (AS) to work correctly in conjunction with an Ingres database, the AS must be built with the Ingres DTP client libraries. The following diagram shows the steps involved:



As shown in the diagram, there are two steps to creating an application server:

1. Precompile the embedded SQL source code.

Invoke the Ingres ESQ precompiler. The precompiler processes your embedded SQL source code and creates a 3GL source code file. For details about precompiling embedded SQL programs, see the *Embedded SQL Companion Guide*.

2. Build the application server.

Issue the Tuxedo buildserver command. You must use the `-r` flag to specify Ingres as the resource manager. For example:

```
buildserver -r INGRES -f server.c -o server -b shm -s SERVICE
```

The example shown compiles the C language source file "server.c", which contains a service named SERVICE, and builds an application server executable named server. For details about the buildserver command, refer to the *Tuxedo Transaction Monitor Reference Manual*.

Step 2: Editing the Application Configuration File

The following section describes entries in the Tuxedo configuration file (“ubbbconfig”) that are relevant to the Ingres DTP for Tuxedo product. Refer to the *Tuxedo Transaction Monitor Administrator’s Guide* for more information.

The *MACHINES Section

The ENVFILE entry in the Tuxedo configuration file enables you to specify a file of environment variables to be set in the application server’s process space. In this file you can specify settings for any of the Ingres environment variables. For a full listing of Ingres environment variables, see the *System Administrator Guide*.

To direct Tuxedo to perform transaction management on behalf of your application by using the XA interface, your application must specify a TLOGDEVICE (a logging file used by Tuxedo for transaction management). For instructions on specifying a TLOGDEVICE, refer to the *Tuxedo Transaction Monitor Administrator’s Guide*.

The *GROUPS Section

The TMS_NAME parameter must be set to the name of the Ingres TMS executable “TMS_INGRES”.

The OPENINFO parameter must be specified as follows:

```
INGRES:[vnodename:]dbname [as connection_name] with tmgroup groupname
[options = flag {, flag}]
```

In the example above, *groupname* is an alphanumeric string of up to 24 characters in length. Within a particular application, the first four characters of the TMGROUP parameter must be unique. For example:

```
*GROUPS
DEFAULT:  TMSNAME=TMS_INGRES      TMSCOUNT=3      LMID=SITE1
BANKB1   GRPNO=1 OPENINFO="INGRES:bankd11 WITH TMGROUP d11bank"
BANKB2   GRPNO=2 OPENINFO="INGRES:bankd12 WITH TMGROUP d12bank"
BANKB3   GRPNO=3 OPENINFO="INGRES:bankd13 WITH TMGROUP d13bank"
```

Step 3: Editing the ENVFILE

The ENVFILE, specified in the Tuxedo configuration file, contains definitions of Ingres environment variables. Environment variables relevant to Ingres DTP for Tuxedo are listed in the following table:

Environment Variable	Description
II_TUXEDO_LOC	Specifies the directory where the shared memory file is to be created. It must be the same for all servers in a group. If II_TUXEDO_LOC is not set, II_TEMPORARY will be used.
II_TUX_SHARED	If set to USER, the name of the shared memory segment used by Ingres will be t<username>.tux; otherwise the name will be t1.tux.
II_TUX_AS_MAX	The maximum number of application and TMS servers that will be started. The default value is 32. A maximum of II_TUX_AS_MAX servers will be permitted to attach to the Ingres shared memory segment.
II_TUX_XN_MAX	The total number of transaction entries allocated in the shared memory segment. Each server that attaches the shared memory segment will reserve II_XA_SESSION_CACHE_LIMIT transaction entries for its own use. The default value is 1024.
II_XA_TRACE_FILE	Specifies a file in which Ingres DTP logs the events occurring through the TMXA interface, as well as any SQL performed against the Ingres DBMS. For more information, see Obtaining Trace and Error Information in the chapter "Troubleshooting and Tuning." The user who starts the application servers must have write access to the file.

All other supported Ingres environment variables (including ING_SET, ING_SYSTEM_SET, and ING_SET_DBNAME) can be set in the ENVFILE.

Starting and Shutting Down Application Servers

Before starting application servers, you must have performed the following tasks:

- Edited the “ubconfig” file to define parameters required for a Tuxedo application (such as UID, GID, IPCKEY, maxgnt, cmtret, and others as required)
- Compiled the “ubconfig” file using the `tmloadcf` command
- Created TLOG devices (using the `tpadmin crdl` command)

To start your application servers, use the `tmboot` command. For details about the preceding requirements, refer to your Tuxedo documentation.

Verifying Server Startup

After a server group has started up, you can use the `tmadmin printserver` command to verify that all servers in a group have started successfully. For each server group accessing an Ingres database, `printserver` will show:

- Two or more TMSs with executables named “TMS_INGRES”
- All the application servers that were configured into the server group

Here is a sample of the `printserver` output:

a.out Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current Service
BBL	9099	SIT	1	0	150	(IDLE)
TLR	00001.00001	BANKB1	1	0	0	(IDLE)
XFER	00001.00004	BANKB1	4	0	0	(IDLE)
ACCT	00001.00007	BANKB1	7	0	0	(IDLE)
BAL	00001.00010	BANKB1	10	0	0	(IDLE)
BTADD	00001.00013	BANKB1	1	0	0	(IDLE)
AUDITC	00001.00016	BANKB1	16	0	0	(IDLE)
BALC	00001.00027	BANKB1	27	0	0	(IDLE)
TMS_INGRES	BANKB1_TMS	BANKB1	30001	0	0	(IDLE)
TMS_INGRES	BANKB1_TMS	BANKB1	30002	0	0	(IDLE)

Application Development Guidelines

The following guidelines will assist you in designing and coding application servers using Ingres DTP for Tuxedo.

Placement of Transaction Demarcation Calls

In coding application servers using Ingres, be aware of the following restrictions:

- If you choose to demarcate transactions in the application server, your transaction must not span multiple server groups. Transactions that span multiple server groups must be demarcated in the application client. (You can place transaction demarcation calls (tpbegin, tpcommit, tpabort) in application clients and application servers.)
- You may define and call AUTOTRAN services in your application, but those services may not make service calls to services in other server groups. That is, your transaction must not span multiple server groups.

Handling Errors

Your application must check for errors after every SQL statement and Tuxedo ATMI call. If your application detects an error, it must abort the current global transaction.

Handling Deadlock

To minimize deadlock between application servers, design your application with the following points in mind:

- Application servers accessing Ingres databases do not share transaction context. Two servers (possibly in the same global transaction) modifying the same page in a table will compete for locks.
- The resource manager (Ingres DBMS) may abort your transaction if it encounters a locking contention problem (for example, deadlock or lock-wait timeout).

The following strategies can help minimize locking problems:

- Place services that access the same table into the same application server process.
- Set the read locking off where appropriate by including the following line in your ENVFILE:

```
ING_SET=SET LOCKMODE SESSION WHERE READLOCK = NOLOCK
```

- Set a loc- wait timeout value in the resource manager to avoid waiting indefinitely for a page or table lock to be freed. Add the following to your ENVFILE:

```
ING_SET=SET LOCKMODE SESSION WHERE TIMEOUT=value
```

where value is $TPBLOCKTIME * SCANUNIT * 0.5$ or less

- Use a hash structure on your table to avoid locking contention on inserts using a sequential key.
- Clients (or services who explicitly initiate transactions) must test the tpcall return status for deadlock or time-out. If either problem is detected, the client should abort the transaction.

For more information on locking strategies, see the *Database Administrator Guide*.

TP_COMMIT_CONTROL

Use of this Tuxedo feature with Ingres DTP is not recommended.

Index

A

application client, described, 2-3
application program, described, 2-2
application server
 CICS/6000, A-7
 described, 2-3
 Encina, B-3
 registering databases, 3-2
 Tuxedo, C-4

C

CICS/6000 applications, A-1
COBOL, CICS/6000 support, A-5
connection names, 3-2, A-8, B-3

D

database procedures
 error handling, 3-4
DCE, configuring with Encina, B-2

E

Encina applications, B-1
error listing files, 4-1

G

global transactions
 aborting, 4-4
 errors, 3-4

two phase commit, 3-7
X/Open DTP standard, 2-1

I

Ingres DTP
 building applications, 3-2
 described, 2-1
installation requirements, 2-5

L

lartool utility, 4-4
log file, 4-1
logging, Encina, B-5
logstat utility, 4-1

M

makefile
 CICS/6000 C applications, A-7
 CICS/6000 COBOL applications, A-8
multiple sessions, 3-2

O

open string, 3-1

R

resource manager definition file, C-3
resource manager instances
 CICS/6000, A-8
 Encina, B-3

restrictions

- multiple sessions, 3-6

- SQL statements in Ingres DTP, 3-4

- Tuxedo application servers, C-8

- two phase commit, 3-7

rollback, transaction, 3-4

S

set connection statement, A-8, B-3

T

TMS Server, C-3

trace file, 4-1, B-5

TRAN-C, B-5

transaction manager, described, 2-1

Tuxedo

- applications, C-1

- configuration file, C-5

X

X/Open DTP standard

- diagram of logical components, 2-3