



PUBLIC

SAP Adaptive Server Enterprise 16.0 SP04

Document Version: 1.0 – 2020-12-10

XA Interface Integration Guide for CICS, Encina, and TUXEDO

Content

- 1 The SAP XA Environment. 4**
- 1.1 Definitions. 4
- 1.2 Overview of the X/Open DTP Model. 5
 - Components of the Model. 6
 - How the Components Communicate. 6
 - How the Components Interact. 7
 - Recovery. 8
- 1.3 The SAP XA Environment. 8
 - Components of the SAP XA Environment. 9
- 1.4 Connections in the SAP XA Environment. 9
 - Identifying Connections Via LRMs. 9
 - Establishing Connections. 11
 - Distributing Work Across LRMs. 12
- 2 Configuring the XA Environment. 13**
- 2.1 Configuring SAP ASE. 13
- 2.2 Open String Parameters for the DTM XA Interface. 13
 - Log file and Trace Flag Parameters. 14
 - xa_open() Function Behavior. 15
- 2.3 XA Configuration File for DTM XA Interface. 16
 - Environment Variable for Specifying the Configuration File. 16
 - [all] Section for Defining Common LRM Parameters. 16
 - Editing the XA Configuration File. 17
 - Additional Capabilities, Properties, and Options. 19
- 2.4 Using the DTM XA Interface with CICS. 21
 - Building the Switch-Load File. 21
 - Adding a SAP Stanza to the CICS Region XAD Definition. 24
- 2.5 Using the DTM XA Interface with Encina. 25
 - Assigning an Open String with monadmin create rm 25
 - Initializing LRMs with mon_RegisterRmi. 26
 - Linking Applications with DTM XA Interface Libraries. 27
 - Establishing Connections. 27
- 2.6 Using the DTM XA Interface with TUXEDO. 27
 - Linking. 28
 - Setting up the UBBCONFIG File. 29
 - Creating the TUXEDO Configuration File. 31
 - Building the TMS. 31

	Building the COBOL Runtime Environment.	32
3	Application Programming Guidelines.	33
3.1	X/Open DTP Versus Traditional SAP Transaction Processing.	33
3.2	Managing Transactions and Connections	33
	Managing Transactions.	34
	Managing Connections.	34
	The Current Connection.	35
	Nontransactional Connections.	35
3.3	Deallocating Cursor Function with Client-Library.	36
3.4	Dynamic SQL.	36
3.5	Obtaining a Client-Library Connection Handle.	36
3.6	Multiple-thread Environment Issues.	38
	Caveats of Thread Use.	39
	Embedded SQL Thread-Safe Code.	39
	Tightly Coupled Transactions.	40
3.7	Linking with CT Library.	41
3.8	Sample Embedded SQL COBOL Fragment.	41
3.9	Sample Embedded SQL C Fragment.	43

1 The SAP XA Environment

The SAP XA environment includes the X/Open DTP model, the components of the SAP XA environment—including the DTM XA Interface, your application program, and SAP Adaptive Server Enterprise (ASE), among others.

This guide serves as a reference manual for:

- System administrators setting up a distributed transaction processing (DTP) environment that includes one or more Adaptive Servers with distributed transaction management features, accessed by transactions from within a CICS, Encina, or TUXEDO transaction manager (TM) system.
- Application programmers using Embedded SQL™ or Client-Library™ to access data on one or more Adaptive Servers.

This manual assumes the reader is familiar with:

- The TM operating environment
- Embedded SQL
- Open Client Client-Library
- Adaptive Server administration

1.1 Definitions

The X/Open DTP model assumes an understanding of certain terms.

- transaction – a whole unit of work consisting of one or more computational tasks. Most often, a transaction's tasks manipulate shared resources.
- committed transaction – a completed transaction whose changes to any shared resources are permanent.
- rolled-back transaction – a complete transaction whose changes to any shared resources are nullified.
- ACID test – the test of a true transaction; to pass, the transaction must exhibit the following properties:
 - Atomicity – all or none of the results of the transaction take effect.
 - Consistency – if a transaction is rolled back, all resources that the transaction affected return to the state they were in prior to the transaction's execution.
 - Isolation – a transaction's results are visible only to that transaction until the transaction commits.
 - Durability – permanent resource changes resulting from commitment survive subsequent system failures.
- transaction processing – a system of coordinating the transactions that multiple users perform on shared, centralized resources.
- distributed transaction processing – a transaction processing model in which the shared resources are located at distinct physical sites on a computer network.
- local transaction – a transaction that affects data in a single database and whose tasks a single resource manager performs. See *Overview of the X/Open DTP Model* for a definition of resource managers.
- global transaction – a transaction that spans more than one database and multiple resource managers.

- transaction branch – a portion of the work that makes up a global transaction.
- transaction identifier – an identifier that a TM assigns to a transaction. The transaction monitor uses the transaction identifier to coordinate all activity related to a global transaction. The resource manager uses the global identifier to match the recoverable tasks it performed for the transaction.
- recovery – the process of bringing a transaction processing system into a consistent state after a failure. Specifically, this means resolving transactions left in a noncommitted state.

Related Information

[Overview of the X/Open DTP Model \[page 5\]](#)

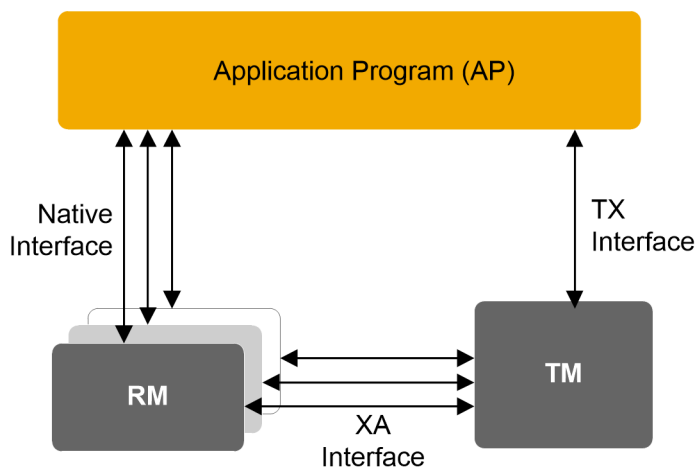
1.2 Overview of the X/Open DTP Model

The X/Open DTP model is a model for software architecture that allows multiple application programs to share resources provided by multiple resource managers, and allows their work to be coordinated into global transactions.

The X/Open DTP model identifies the key entities in a distributed transaction processing environment and standardizes their roles and interactions. The entities are:

- The transaction processing monitor (TM)
- The resource manager (RM)
- The application program (AP)

This diagram, shows the X/Open DTP functional model, including its major components and their interfaces.



These components communicate through the native, XA, and TX Interfaces.

Related Information

[How the Components Communicate \[page 6\]](#)

1.2.1 Components of the Model

The X/Open DTP functional model consists of the components: application program (AP), resource manager (RM), and transaction processing monitor (TM).

The AP contains the code written to accomplish a particular transaction or portion thereof. As such, it designates the beginning and end of global transactions.

The RM provides access to shared resources. Database servers, file servers, and print servers are examples of RMs. In a typical X/Open DTP environment, a single AP communicates with more than one RM. In the SAP XA environment, the RM is an SAP® ASE database.

The TM coordinates the communication between all parties participating in the transaction. The TM assures that the work done by the AP is contained in a global transaction, which commits or aborts atomically.

Specifically, the TM's tasks include:

- Assigning global identifiers to transactions.
- Monitoring the progress of global transactions.
- Coordinating the flow of transaction information between the APs and the RMs.
- Managing the transaction commitment protocol and failure recovery.

Related Information

[How the Components Interact \[page 7\]](#)

1.2.2 How the Components Communicate

The AP, the RM, and the TM communicate via three distinct interfaces: native, TX, and XA.

The native interface is the medium by which the AP makes requests directly to the RM. This interface is RM specific. In the SAP XA environment, the native interface is either Embedded SQL or Client-Library.

The TX Interface is the medium between the AP and the TM. The AP uses TX calls to delineate transaction boundaries. In other words, the AP requests that the TM start and commit or roll back global transactions, via the TX Interface. This interface is TM specific.

The XA Interface is the medium between the RM and the TM. The DTM XA Interface is the SAP version of the interface for SAP ASE. Using XA calls, the TM tells the RM when transactions start, commit, and roll back. The TM also handles recovery.

1.2.3 How the Components Interact

The components work together to process transactions from initiation through completion.

The AP delimits transaction boundaries. An AP informs the TM, via TX calls, that a global transaction is beginning. The TM then communicates with all available RMs, via XA calls, to associate a single transaction identifier with any work the RMs will do on behalf of the AP within the bounds of the global transaction.

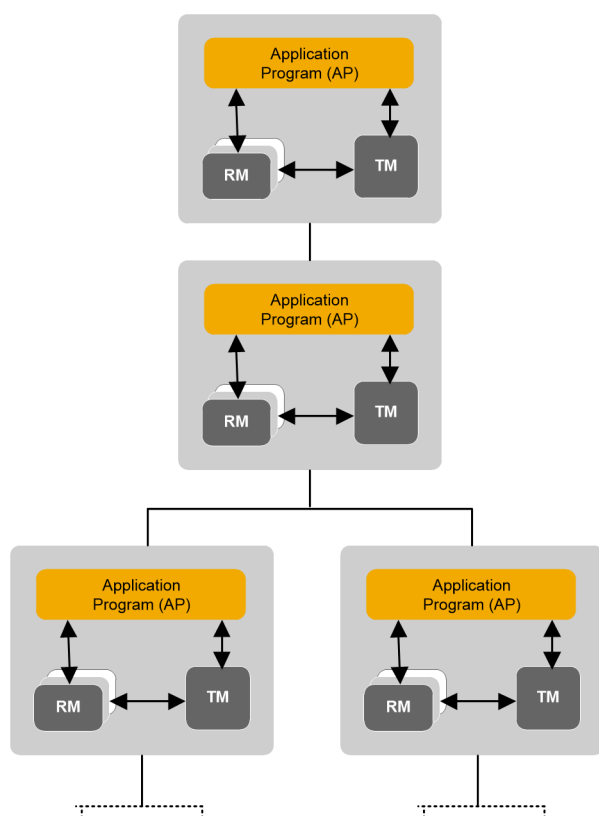
When the AP requests that the TM commit the global transaction, the TM and the RMs use the two-phase commit protocol to guarantee transaction atomicity.

Transaction completion takes place in two phases—the prepare phase and the commit phase. For a detailed description of the two-phase commit protocol, see the *Open Client DB-Library/C Reference Manual*.

In the prepare phase, the TM requests each RM to prepare to commit its portion of the global transaction. This portion is known as a transaction branch.

In the commit phase, the TM instructs the RMs to commit or abort their branches of the transaction. If all RMs report back that they have prepared their respective transaction branches, the TM commits the entire transaction. If any RM reports that it was unprepared or fails to respond, the TM rolls back the entire transaction.

This diagram shows a typical transaction branch structure.



1.2.4 Recovery

The TM is responsible for managing global recovery. In certain situations, an administrator may decide to complete its transaction branch independently of the TM.

When this occurs, the administrator's decision is called a heuristic decision.

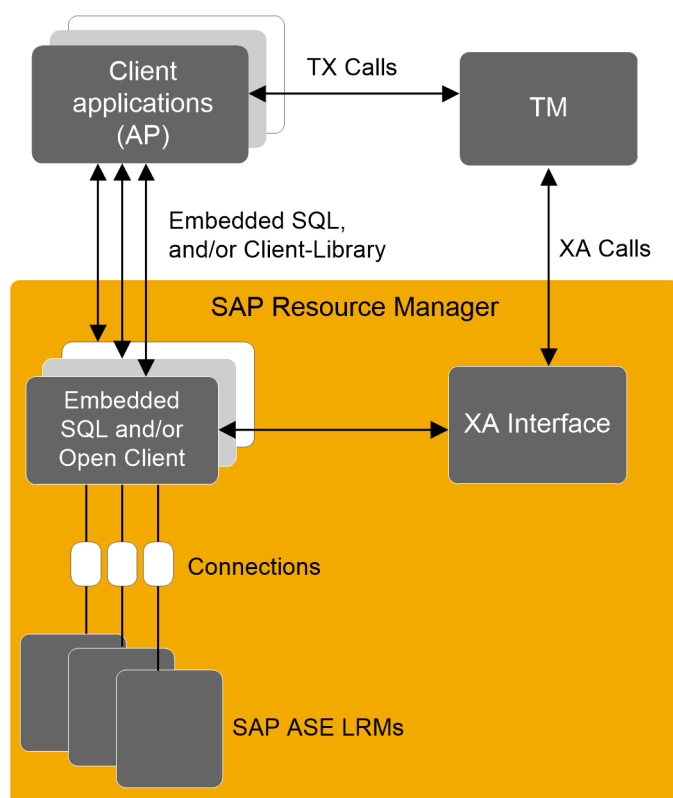
The heuristic decision may be in conflict with the TM's decision. For example, the administrator may commit a transaction branch and the TM may request to abort it.

Such a conflict requires manual intervention from the System Administrator. For a discussion of heuristic decisions in the SAP XA environment, see the *System Administration Guide: Volume 2*.

1.3 The SAP XA Environment

The DTM XA Interface relies on the SAP transaction processing model to implement X/Open's DTP model.

SAP ASE is used as an RM, as shown in this diagram of the XA DTP model.



1.3.1 Components of the SAP XA Environment

The SAP XA environment is made up of several components.

- SAP DTM XA Interface– this is the SAP implementation of the XA Interface for SAP ASE, described in *How the Components Communicate*.
- The Open Client libraries – Client-Library calls can be part of the native interface between your application and the resource manager.

i Note

XA Client libraries run in 32-bit mode on Solaris 64, AIX 64, HP-UX 64 and Windows platforms and in 64-bit mode on Solaris 64, AIX 64, HP-UX 64 platforms.

- Embedded SQL/C and Embedded SQL/COBOL – embedded SQL calls can be part of the native interface between your application and the resource manager.
- One or more SAP ASE servers– these play the role of RMs.
- The XA configuration file– this file contains entries that define client/server connections for use with XA.
- A set of XA-specific `dbcc` commands – system Administrators use these to manage heuristic transactions.
- TM- configuration files and commands – this configuration and the commands are specific to TM.

Configuring the XA Environment explains how to configure these components so that transactions can use the DTM XA Interface to access data stored on SAP ASE.

Related Information

[How the Components Communicate \[page 6\]](#)

[Configuring the XA Environment \[page 13\]](#)

1.4 Connections in the SAP XA Environment

The X/Open DTP model has no notion of connections, yet connections are central to the SAP client/server architecture. The SAP XA environment must resolve this discrepancy.

To this end, the SAP XA environment introduces the notion of a logical resource manager (LRM).

1.4.1 Identifying Connections Via LRMs

Each instance of the SAP RM appears to the TM as one or more LRMs.

An LRM associates a symbolic name with a client/server connection. An AP uses the names to identify the specific physical connection to one or more SAP ASE servers. The TM uses the names to open connections on behalf of the AP.

1.4.1.1 Connection Information Storage

The following components of the SAP XA environment contain information about LRMs. The System Administrator configures these files before starting up the TM.

For information on the full configuration process, see *Configuring the XA Environment*.

The SAP XA Configuration File

The SAP XA configuration file contains one entry per LRM. The entry associates the LRM with a physical SAP ASE name, and assigns pre-connection Client-Library capabilities and properties to the LRM. For details on the XA configuration file, see *XA Configuration File for DTM XA Interface*.

The CICS XA Product Definition (XAD)

The CICS XAD contains one stanza per LRM. The stanza assigns each LRM a user name and password in the form of an open string. The user name and password enable the SAP XA environment to control a particular connection's access to SAP ASE resources. For details on the CICS XAD file, see *Adding a SAP Stanza to the CICS Region XAD Definition*.

The Encina `monadmin create rm` Command

The `monadmin create rm` command assigns each LRM a user name and password in the form of an open string. The user name and password allow the SAP XA environment to control a particular connection's access to SAP ASE resources. For details on the Encina `monadmin` command, see *Assigning an Open String with `monadmin create rm`*. Your current version of Encina may have additional commands for specifying RMs.

i Note

You can use the Encina `enconsole` interactive command instead of the shell `monadmin` command.

For detailed information, see the *Encina Monitor System Administrator's Guide and Reference*.

TUXEDO's UBBCONFIG File

In addition to modifying the SAP configuration files, integrating TUXEDO requires customizing the TUXEDO configuration file, `UBBCONFIG`. The open string is the only portion of the `UBBCONFIG` file that requires modification. It includes the user name and password, which allow XA-Server to control a connection's access to SAP ASE resources. See *Setting up the UBBCONFIG File* for details.

Related Information

[Configuring the XA Environment \[page 13\]](#)

[XA Configuration File for DTM XA Interface \[page 16\]](#)

[Adding a SAP Stanza to the CICS Region XAD Definition \[page 24\]](#)

[Assigning an Open String with `monadmin create rm` \[page 25\]](#)

[Setting up the UBBCONFIG File \[page 29\]](#)

1.4.2 Establishing Connections

The TM, together with the XA Interface, establishes connections between applications and RMs in several steps.

Connections in a CICS Environment

1. When the CICS region starts up, it issues an XA open call to each LRM configured in an XAD, using the information contained in each open string.
2. The CICS region passes to the XA Interface library the open string associated with each stanza. The open string contains the LRM name, the user name, and the password.
3. The XA Interface looks up the LRM name in the SAP XA configuration file and matches it to an actual RM name, that is, an actual physical SAP ASE server. The RM name matches an entry in the SAP ASE `interfaces` file.
4. The XA Interface establishes one connection to an SAP ASE server for each LRM entry. The XA Interface confers on any connection the pre-connection properties and capabilities configured for the LRM.

Connections in an Encina Environment

1. An application issues a `mon_RegisterRmi` function, thereby requesting use of an LRM.
2. Using information contained in an open string, the TM issues an XA open call to the LRM (configured in the `monadmin create rm` command) whose name matches that issued in step 1, above.
3. The XA Interface looks up the LRM name in the SAP XA configuration file and matches it to an actual RM name—that is, to an actual physical SAP ASE server. The RM name matches an entry in the SAP ASE `interfaces` file.
4. The XA Interface establishes one logical connection to an SAP ASE server for each LRM entry. The XA Interface confers on any connection the pre-connection properties and capabilities configured for the LRM.

Connections in a TUXEDO Environment

1. The application uses the LRM specified in the `UBBCONFIG` file to reference the logical connection for a branch of a global transaction. In using the LRM name, the application implicitly requests and establishes an LRM.
2. The transaction manager passes the appropriate open string to the XA Interface through the LRM whose name matches the one issued in step 1. The XA Interface uses the LRM name, the user name, and the password.
3. The XA Interface looks in the `xa_config` file to find an association between the LRM name and SAP ASE. The server name matches an entry in the `interfaces` file where its network information is kept.
4. The XA Interface establishes one logical connection to a server for each LRM entry. The XA Interface confers on any connection the pre-connection properties and capabilities configured for the LRM.

1.4.3 Distributing Work Across LRMs

The System Administrator and the application programmer together must agree on the number and names of LRMs that their SAP XA environment includes.

The System Administrator configures the TM and SAP XA configuration files accordingly. The application programmer invokes a particular LRM name within the application code to send a portion of a global transaction across that connection. The TM coordinates this distribution.

You can configure the SAP XA environment for more connections than are actually used. That is, the XA configuration file may contain inactive entries.

For example, in a CICS environment with an SYBXA_1, and an LRM connection `connection_1`,

- The `interfaces` file includes server information:

```
sybXA_1 query tcp ether groucho 6161
```

- The XA Configuration file includes connection and server information:

```
; lrm - Names the logical connection as seen by the application and the TP
monitor.
; server - Names the physical server as found in the SAP interfaces file.
lrm=connection_1
server=SYBXA_1
```

- The XAD File stanza for the LRM contains connection information:

```
XAOpen="-User1 -Ppassword1 -Nconnection_1"
```

- The application program contains transactions using `connection_1`.

2 Configuring the XA Environment

You must link the DTM XA Interface library with your X/Open XA-compliant transaction manager to use SAP ASE as a resource manager.

i Note

See the `README` file under the subdirectories of `$$SYBASE/$$SYBASE_OCS/sample` for detailed information about configuring the DTM XA Interface for your system.

2.1 Configuring SAP ASE

Your SAP ASE server must be configured to use the distributed transaction management feature to function in a SAP XA environment.

Enable distributed transaction management with the `enable dtm` configuration parameter:

```
sp_configure 'enable dtm', 1
```

You must restart SAP ASE for this parameter to take effect.

See the *System Administration Guide: Volume 2 > Distributed Transaction Management* for more information.

2.2 Open String Parameters for the DTM XA Interface

The X/Open XA specification allows each resource manager vendor to define an open string and a close string. The DTM XA Interface does not require or use the close string.

The DTM XA Interface uses the required and optional open string parameters described below.

The format for parameters in the open string for the DTM XA Interface is:

```
-N<lrn_name> -U<username> -P<password >[-L<logfile_name>]  
[-T<traceflags>] [-V11] [-O1] [-O-1]
```

Parameter	Meaning
<lrn_name>	The name of the LRM as defined in the XA configuration file.

Parameter	Meaning
<username>	The user name used to log in to SAP ASE. In the open string for resource managers, the specified <username> must have the dtm_tm_role in the corresponding SAP ASE server. The System Security Officer can assign this role using sp_role or the grant command.
<password>	The password accompanying the user name.
<logfile_name>	The fully qualified file name to which the XA Interface writes tracing information (optional). The XA Interface initializes the log file and trace flag settings with the initial xa_open() call. If no <logfile_name> is specified, then the DTM XA Interface logs information to a file named syb_xa_log in the current directory.
<traceflags>	Trace flags control the output that is written to the log file (optional). See <i>Section for Defining Common LRM Parameters</i> for a list of valid trace flags.
-V11	Specifies Open Client version 11 behavior for backward compatibility (optional).
-O1 or -O-1	Specifies an option for transaction operation. At this time only -O1 (tightly coupled transaction branches) is supported. -O1 sets the option. -O-1 clears the option. -O-1 is the default.

⚠ Caution
See *Tightly Coupled Transactions* before setting -O1.

Related Information

[\[all\] Section for Defining Common LRM Parameters \[page 16\]](#)
[Tightly Coupled Transactions \[page 40\]](#)

2.2.1 Log file and Trace Flag Parameters

With the DTM XA Interface to SAP ASE, log file and trace flag parameters can be defined in the [all] section of the XA configuration file, rather than in the X/Open XA open string.

See *Section for Defining Common LRM Parameters* for more information about the log file and trace flag components.

The DTM XA Interface marks each entry in the log file with a label indicating the severity or cause of the message. describes each label.

Label	Type of Entry
Error	An error returned to the transaction manager

Label	Type of Entry
Fatal Error	A severe failure in the DTM XA Interface, with a possible application or transaction manager error
Message	Additional information about a previous error, or a description of the operational environment
Warning	A condition that may indicate problems with the transactional system
Note	Information that does not indicate a problem, but may be useful if an error occurs
XA trace	Information logged as a result of the xa trace flag setting
RM trace	Information logged as a result of the xl trace flag setting
Connection trace	Information logged as a result of the xc trace flag setting
ASE I/F trace	Information logged as a result of the xs trace flag setting
Misc trace	Information logged as a result of the misc trace flag setting
Event trace	Information logged as a result of the event trace flag setting
Verbose trace	Information logged as a result of the v trace flag setting
Function trace	Information logged as a result of the cmn trace flag setting
Open Client trace	Information logged as a result of the ct trace flag setting

Related Information

[\[all\] Section for Defining Common LRM Parameters \[page 16\]](#)

2.2.2 xa_open() Function Behavior

The X/Open XA function, `xa_open()`, initiates a single connection to SAP ASE.

The `<username>` and `<password>` defined in the open string must possess the `dtm_tm_role` in the server. The System Security Officer can assign this role using `sp_role` or the `grant` command. For example:

```
sp_role "grant", dtm_tm_role, user_name
```

2.3 XA Configuration File for DTM XA Interface

The DTM XA Interface to SAP ASE provides mechanisms for configuring the Open Client connections using the XA configuration file. Set all connection capabilities, properties, and options using the XA configuration file.

2.3.1 Environment Variable for Specifying the Configuration File

The DTM XA Interface uses the environment variable XACONFIGFILE to find the full path and file name of the XA configuration file.

You can set this environment variable to specify different locations and names to use for configuration information as necessary.

For example, on UNIX platforms:

```
setenv XACONFIGFILE /usr/u/sybase/xaconfig1.txt
```

If XACONFIGFILE is not defined, or if it does not specify a valid configuration file, the DTM XA Interface looks for a file named `xa_config` in the following directories:

- `$$SYBASE/$$SYBASE_OCS/config`
- `$$SYBASE/$$SYBASE_OCS`
- `$$SYBASE/config`
- `$$SYBASE`

The DTM XA Interface uses the first `xa_config` file it finds.

2.3.2 [all] Section for Defining Common LRM Parameters

The DTM XA Interface uses the [all] section to define parameters that apply to all LRMs.

Certain parameters defined in the [all] section—log file and trace flag definitions—may also be defined in the open string for X/Open XA transaction managers.

Entries for the [all] section in the XA configuration file are as follows:

```
[all]
logfile="<logfile_name>"
traceflags="[xa | xl | xc | cm| event | misc | os | ct | all]"
[properties=<name>=<value>] [...]
```

i Note

The "0x" values in the `xa_config` file can be lost or converted to non-ASCII characters because of the way some characters are processed. To avoid this problem, you must use quotes around all string values. Failure to do so can result in the introduction of unexpected characters into the string.

Parameter	Meaning
<code><logfile_name></code>	The fully qualified file name to which the DTM XA Interface writes tracing information. The DTM XA Interface initializes the log file and trace flag settings with the initial <code>xa_open()</code> call.

<code><traceflags></code>	The trace flags control the output that is written to the log file. Specify one or more of the following flags: <ul style="list-style-type: none"> • <code>all</code> – all tracing. • <code>ct</code> – the <code>ct_debug</code> option with the <code>CS_DBG_ERROR</code> flag (<code>ct_debug</code> functionality is available only from within the debug version of Client-Library). • <code>cmn</code> – entry and exit point tracing of internal XA Interface functions. • <code>event</code> – tracing of significant internal events. • <code>misc</code> – tracing of activities and information for problem resolution. • <code>xa</code> – entry and exit point tracing at the <code>xa_*</code> level. • <code>xc</code> – entry and exit point tracing at the <code>xc_*</code> level. • <code>xl</code> – entry and exit point tracing at the <code>xl_*</code> level.
---------------------------------	--

i Note

Tracing at the `xc_*`, `xl_*`, `event`, `misc`, and `cmn` levels is intended to be meaningful only to SAP development. Specify these tracing levels only when instructed to do so by SAP Product Support.

<code><properties></code>	i Note The following property must be set in the <code>[all]</code> stanza. You cannot set it in the <code>[xa]</code> stanza: <code>CS_LOGIN_TIMEOUT=<timeout></code>
---------------------------------	---

You can define these optional properties in the `[all]` section of the XA configuration file:

- `PROPERTIES=CS_DISABLE_POLL=[CS_TRUE | CS_FALSE]`
- `PROPERTIES=CS_EXTRA_INF=[CS_TRUE | CS_FALSE]`
- `PROPERTIES=CS_HIDDEN_KEYS=[CS_TRUE | CS_FALSE]`
- `PROPERTIES=CS_MAX_CONNECT=<number_of_connections>`
- `PROPERTIES=CS_NOINTERRUPT=[CS_TRUE | CS_FALSE]`
- `PROPERTIES=CS_TEXTLIMIT=<textlimit>`
- `PROPERTIES=CS_TIMEOUT=<timeout>`

2.3.3 Editing the XA Configuration File

You must customize the XA configuration file for the application environment. Use the text editor of your choice to open the XA configuration for editing.

The sample contents of an XA configuration file are as follows:

```
; Comment line as first line of file REQUIRED!
;
; xa_config - sample xa_config file.
```

```

;
; Note that the Adaptive Server names may need
; to be customized for your environment.
; simprpc.ct sample application entry.
[all]
    logfile="<logfile_name>"
    traceflags="<traceflags>"
    properties="<name>"="<value>" [, "<name>"="<value>"] [...]<
>
[xa]
    lrm="<connection1>"
    server="<sqlserver>"

; Rentapp sample xa_config entries.

[xa]
    lrm="FLEET_CON"
    server="fleetsrv"

[xa]
    lrm="RESERVE_CON"
    server="rsrvsrv"

```

Note

The first line of the `xa_config` file *must* be a comment that is denoted by a semicolon (;) in the first character position.

For each additional LRM, create an entry with the following format. Keep the `connection1` entry for installation verification.

```

[xa]
<tab> lrm="<connection_name>"
<tab> server="<adaptive_server_name>"
<tab> capabilities="<name>"="<value>" [, "<name>"="<value>"] [...]
<tab> properties="<name>"="<value>" [, "<name>"="<value>"] [...]
<tab> options="<name>"="<value>" [, "<name>"="<value>"] [...]

```

The `<connection_name>` is the symbolic name for the connection between the application and SQL. The `<adaptive_server_name>` is the name of the SAP ASE server associated with the connection. `<adaptive_server_name>` must correspond to a server name defined in the `interfaces` file.

See *Additional Capabilities, Properties, and Options* for information a list of capabilities, properties, and options that can be used with the DTM XA Interface.

Related Information

[Additional Capabilities, Properties, and Options \[page 19\]](#)

2.3.4 Additional Capabilities, Properties, and Options

XA configuration file entries for capabilities, properties, and options have a general format. All names and values for these capabilities, properties, and options correspond to CS-Library keywords.

```
<tab> capabilities="<name>"="<value>" [, "<name>"="<value>"] [...]  
<tab> properties="<name>"="<value>" [, "<name>"="<value>"] [...]  
<tab> options="<name>"="<value>" [, "<name>"="<value>"] [...]
```

The following is the list of the names for capabilities that can be defined in the XA configuration file for the DTM XA Interface. Unless otherwise specified in these tables, the valid values for each capability is CS_TRUE or CS_FALSE.

Note

See the *Open Client Client-Library/C Reference Manual* for specific descriptions.

CS_CON_NOINBAND	CS_DATA_NODATETIMEN	CS_DATA_NOMONEYN
CS_CON_NOOOB	CS_DATA_NODEC	CS_DATA_NONUM
CS_DATA_NOBIN	CS_DATA_NOFLT4	CS_DATA_NOSENSITIVITY
CS_DATA_NOVBIN	CS_DATA_NOFLT8	CS_DATA_NOTEXT
CS_DATA_NOLBIN	CS_DATA_NOIMAGE	CS_PROTO_NOBULK
CS_DATA_NOBIT	CS_DATA_NOINT1	CS_PROTO_NOTEXT
CS_DATA_NOBOUNDARY	CS_DATA_NOINT2	CS_RES_NOEED
CS_DATA_NOCHAR	CS_DATA_NOINT4	CS_RES_NOMSG
CS_DATA_NOVCHAR	CS_DATA_NOINT8	CS_RES_NOPARAM
CS_DATA_NOLCHAR	CS_DATA_NOINTN	CS_RES_NOTDSDEBUG
CS_DATA_NODATE4	CS_DATA_NOMNY4	CS_RES_NOSTRIPBLANKS
CS_DATA_NODATE8	CS_DATA_NOMNY8	

The following lists the names for properties that can be defined in the XA configuration file for the DTM XA Interface. Unless otherwise specified in these tables, the valid values for each property is CS_TRUE or CS_FALSE.

CS_ASYNC_NOTIFS	CS_SEC_NEGOTIATE
CS_DIAG_TIMEOUT	CS_TDS_VERSION= [CS_TDS_40 CS_TDS_42 CS_TDS_46 CS_TDS_50]

CS_DISABLE_POLL	CS_TEXTLIMIT=<textlimit>
CS_HIDDEN_KEYS	CS_EXTRA_INF
CS_PACKETSIZE=<packetsize>	CS_MAX_CONNECT=<connections>
CS_SEC_APPDEFINED	CS_NOINTERRUPT
CS_SEC_CHALLENGE	CS_TIMEOUT=<timeout>
CS_SEC_ENCRYPTION	

The following lists the names for options that can be defined in the XA configuration file for the DTM XA Interface. Unless otherwise specified in these tables, the valid values for each option is CS_TRUE or CS_FALSE.

CS_OPT_ANSINULL
CS_OPT_ANSIPERM
CS_OPT_ARITHABORT
CS_OPT_ARITHIGNORE
CS_OPT_DATEFIRST=[CS_OPT_SUNDAY CS_OPT_MONDAY CS_OPT_TUESDAY CS_OPT_WEDNESDAY CS_OPT_THURSDAY CS_OPT_FRIDAY CS_OPT_SATURDAY]
CS_OPT_DATEFORMAT=[CS_OPT_FMTMDY CS_OPT_FMTDMY CS_OPT_FMTYMD CS_OPT_FMTYDM CS_OPT_FMTMYD CS_OPT_FMTDYM]
CS_OPT_FIPSFLAG
CS_OPT_FORCEPLAN
CS_OPT_FORMATONLY
CS_OPT_GETDATA
CS_OPT_ISOLATION=[CS_OPT_LEVEL1 CS_OPT_LEVEL3]
CS_OPT_NOCOUNT
CS_OPT_NOEXEC
CS_OPT_PARSEONLY
CS_OPT_QUOTED_IDENT
CS_OPT_RESTREES
CS_OPT_ROWCOUNT=<rowcount>

```
CS_OPT_SHOWPLAN
```

```
CS_OPT_STATS_IO
```

```
CS_OPT_STATS_TIME
```

```
CS_OPT_STR_RTRUNC
```

```
CS_OPT_TEXTSIZE=<textsize>
```

2.4 Using the DTM XA Interface with CICS

Setup your CICS environment to use the DTM XA Interface.

See also *XA Configuration File for DTM XA Interface* for information on creating an XA configuration file.

Related Information

[XA Configuration File for DTM XA Interface \[page 16\]](#)

2.4.1 Building the Switch-Load File

Each RM defined in the CICS environment must provide an XA switch-load file. The switch-load file is a component of your CICS configuration; it is referenced in the XAD.

It contains the RM's name, a flag, a version number and a set of non-null pointers to the RM's entry points, provided by the DTM XA Interface.

All of the SAP XADs share a single switch-load file. You can build your SAP switch-load file using the file `sybasexa.c`, which is located in:

```
$$SYBASE/$SYBASE_OCS/sample/xa-dtm/cics/switch
```

The following is a listing of `sybasexa.c`:

```
/*
**
** sybasexa.c
**
** The sybasexa routine references the Sybase xa
** switch structure named "sybase_TXS_xa_switch".
** The switch structure is part of the
** XA product library "libdtmxa.a".
**
** See your CICS documentation for details on the
** switch-load file.
```

```

*/
#include <stdio.h>
#include <tmxa/xa.h>

extern struct xa_switch_t sybase_TXS_xa_switch;
extern struct xa_switch_t RegXA_xa_switch;
extern struct xa_switch_t *cics_xa_switch;

struct xa_switch_t *sybasexa(void)
{
    cics_xa_switch = &sybase_TXS_xa_switch;

    cics_xa_init();

    return(&RegXA_xa_switch);
}

```

This source code references the SAP XA switch structure, which is global to the DTM XA Interface and defined as follows:

```

struct xa_switch_t sybase_TXS_xa_switch =
{
    "SYBASE_SQL_SERVER",
    TMNOFLAGS,
    0,
    xa_open,
    xa_close,
    xa_start,
    xa_end,
    xa_rollback,
    xa_prepare,
    xa_commit,
    xa_recover,
    xa_forget,
    xa_complete
};

```

The use of `TMNOFLAGS` specifies that the DTM XA Interface supports thread migration but does not support dynamic registration or asynchronous operations. For a description of these features, see the *X/Open CAE Specification (December 1991) Distributed Transaction Processing: The XA Specification*.

2.4.1.1 Compiling the Switch-Load File on IBM RISC System/6000 AIX

Compile `sybasexa.c` using the makefile `sybasexa.mk`, which is located in `$$SYBASE/$$SYBASE_OCS/sample/xa-dtm/cics/switch`.

This is a listing of `sybasexa.mk`. Edit it to reflect your configuration:

```

SYB_LIBDIR = $(SYBASE)/$(SYBASE_OCS)/lib
SYB_LIBS = -lxadtm -lct_r.so -lcs_r.so -ltcl_r.so -lcomn_r.so -lintl_r
           -lxdxom
all : sybasexa.c xlc_r4 -bnoquiet -v -D_THREAD_SAFE \
      -I/usr/lpp/encina/include sybasexa.c \
      -o sybasexa \
      -esybasexa \
      -L/usr/lpp/cics/lib \
      -L$(SYB_LIBDIR) \

```

```
$(SYBLIBS) \  
-lcicsrt -ldce -lm \  
/usr/lpp/cics/lib/regxa_swxa.o
```

i Note

You must use the shareable versions of CS-Library (`libcs_r.so.`) and Common Library (`libcom_r.so.`).

2.4.1.2 Compiling the Switch-Load File on HP9000 Series 800 HP-UX

Compile `sybasexa.c` using the makefile `sybasexa.mk.hp800`, which is located in `$(SYBASE)/$(SYBASE_OCS)/sample/xa-dtm/cics/switch`.

This is a listing of `sybasexa.mk.hp800`. Edit it to reflect your configuration.

```
#  
# Makefile to compile the LoadSwitchTable  
# This makefile should be run with the command  
# "make -f sybasexa.mk.hp800"  
#  
CC=/opt/ansic/bin/cc  
CCOPTS= -Aa +z -Dsybasexa=CICS_XA_Init  
ENCINA=/opt/encina  
CICS=/opt/cics  
LD=/usr/ccs/bin/ld  
SYB_LIBDIR = $(SYBASE)/$(SYBASE_OCS)/lib  
CICS_LIBDIR = $(CICS)/lib  
all: sybasexa  
sybasexa: sybasexa.o  
$(LD) -b \  
+e CICS_XA_Init \  
-o sybasexa \  
sybasexa.o \  
$(CICS_LIBDIR)/regxa_swxa.o \  
-Bimmediate -Bnonfatal +s +b/opt/cics/lib \  
$(SYB_LIBDIR)/libxadtm.a \  
$(SYB_LIBDIR)/libct_r.a \  
$(SYB_LIBDIR)/libcs_r.sl \  
$(SYB_LIBDIR)/libtcl_v.a \  
$(SYB_LIBDIR)/libcomn_v.sl \  
$(SYB_LIBDIR)/libintl_r.sl \  
-lm \  
$(CICS_LIBDIR)/libcicsrt.sl \  
-lc  
sybasexa.o: sybasexa.c  
$(CC) -c $(CCOPTS) \  
-I$(ENCINA)/include sybasexa.c
```

i Note

You must use the shareable versions of CS-Library (`libcs_r.sl`) and Common Library (`libcomn_r.sl`).

You must have the ANSI C compiler to build the Load Switch Table.

2.4.1.3 Compiling the Switch-Load File on Sun Solaris 2.x (SPARC)

Compile `sybasexa.c` using the makefile `sybasexa_sol.mk` which is located in `$SYBASE/$SYBASE_OCS/sample_dtm/cics/switch`.

This is a listing of `sybasexa_sol.mk`. Edit it to reflect your configuration.

```
#Makefile to compile the LoadSwitchTable
#This makefile should be run with the command "make -f sybasexa_sol.mk"
SYB_LIBDIR = $(SYBASE)/lib
SYBLIBS = lxatm -lct_r -lcs_r.so -ltcl_r -lcomn_r.so
-lintl_r -lxdxom -lm
all: sybasexa.c
/bin/xlc_r -v -D_THREAD_SAFE \
-I /usr/lpp/encina/include sybasexa.c \
-o sybasexa \
-esybasexa \
-L/usr/lpp/cics/lib \
-L$(SYBLIBS) \
-lcicsrt \
/usr/lpp/cics/lib/regxa_swxa.o -ldce
```

2.4.2 Adding a SAP Stanza to the CICS Region XAD Definition

The CICS TM uses CICS XAD information to communicate with other RMs. The XAD definition contains one SAP stanza for each LRM.

For a description of an XAD stanza's attributes, see your CICS documentation.

Below are two sample SAP XAD stanzas. Use the `SMIT` utility to add stanzas to your CICS region:

```
betaOne:
GroupName=""
ActivateOnStartup=yes
ResourceDescription="XA Product Definition"
AmendCounter=2
Permanent=no
SwitchLoadFile="/usr/lpp/sybase/sample/xa_library/
                cics/switch/sybasexa"
XAOpen="-User_1 -Ppassword_1 -Nconnection_1"
XAClose="ignored"
XASerialize=all_operations
betaTwo:
GroupName=""
ActivateOnStartup=yes
ResourceDescription="XA Product Definition"
AmendCounter=2
Permanent=no
SwitchLoadFile="/usr/lpp/sybase/sample/xa_library/
                cics/switch/sybasexa"
XAOpen="-User_2 -Ppassword_2 -Nconnection_2"
XAClose="ignored"
XASerialize=all_operations
```

The following fields are configuration-dependent and must be modified:

- SwitchLoadFile
- XAOpen
- XAClose
- XASerialize

i Note

All SAP stanzas can use the same switch-load file.

See *Open String Parameters for the DTM XA Interface* for information about the contents specified in the XAOpen string of the XAD Definition.

Related Information

[Open String Parameters for the DTM XA Interface \[page 13\]](#)

2.5 Using the DTM XA Interface with Encina

You can assign an open string and initialize an RM for use with the Encina.

Link applications with the DTM XA Interface library, `libxadtm.a`.

See also *XA configuration file for DTM XA Interface* for information on creating an XA configuration file.

Related Information

[XA Configuration File for DTM XA Interface \[page 16\]](#)

2.5.1 Assigning an Open String with `monadmin create rm`

The `monadmin create rm` command assigns each LRM a user name and password in the form of an *open string*. The user name and password allow the DTM XA Interface to control a particular connection's access to SAP ASE resources.

See *Open String Parameters for the DTM XA Interface* for more information about the contents of the open string.

The following shows sample screen contents of a `monadmin create rm` session:

```
echo "Creating connection_1 resource manager record"
monadmin delete rm connection_1 >>& demo_conf.log
monadmin create rm connection_1\
```

```

-open "-Usa -Psecret -Nconnection_1" \
-close "not used" >>& \
demo_conf.log
if ($status) then
echo "Failed to create lrm_1 resource mgr.";
exit 1;
endif

```

Your current version of Encina may have additional commands for specifying RMs. For detailed information, see the *Encina Monitor System Administrator's Guide and Reference*.

Note

You can use Encina `enconsole` interactive command instead of the shell `monadmin` command.

Related Information

[Open String Parameters for the DTM XA Interface \[page 13\]](#)

2.5.2 Initializing LRMs with `mon_RegisterRmi`

From within your Encina Monitor application server, you must register each LRM with a call to `mon_RegisterRmi`.

For example:

```

status =
mon_RegisterRmi(&sybase_TXS_xa_switch,"connection_1",
&rmiID);
if (status != MON_SUCCESS)
{
fprintf(stderr, "mon_RegisterRmi failed (%s).\n",
mon_StatusToString(status));
bde_Exit(1);
}
fprintf(stderr, "mon_RegisterRmi complete\n");

```

For each LRM registered with a `monadmin create rm` command, there must be a `mon_RegisterRmi` command that initializes the LRM. The `<rmname>` specified in the `monadmin create rm` command must match the `<rmname>` in the `mon_RegisterRmi` command.

See the *Encina Monitor Programmer's Guide* for:

- Information about the tasks performed by the registration function and the order in which they must be performed
- Full syntax of the `mon_RegisterRmi` command

2.5.3 Linking Applications with DTM XA Interface Libraries

Link applications with the DTM XA Interface library, `libxadtm.a`.

2.5.4 Establishing Connections

The TM, together with the DTM XA Interface library, establishes connections between applications and RMs.

1. An application issues a `mon_RegisterRmi` function, thereby requesting use of an LRM.
2. Using information contained in an open string, the TM issues an XA open call to the LRM (configured in the `monadmin create rm` command) whose name matches that issued in step 1, above.
3. The TM passes the open string associated with each `monadmin create rm` command to the DTM XA Interface. The open string contains the LRM name.
4. The DTM XA Interface looks up the LRM name in the XA configuration file and matches it to an actual RM name—that is, to an actual physical SAP ASE server. The RM name matches an entry in the SAP ASE `interfaces` file.
5. The DTM XA Interface establishes one logical connection to a SAP ASE server for each LRM entry. It then confers on any connection the preconnection properties and capabilities configured for the LRM.

2.6 Using the DTM XA Interface with TUXEDO

There are application-specific steps you need to perform to integrate the XA Interface with TUXEDO.

The application-specific part of the integration involves:

- Linking the application with the application servers
- Setting up the `UBBCONFIG` file
- Building a transaction monitor server (TMS)
- Integrating the application servers with the resource managers

It is assumed that TUXEDO is installed in the `$TUXDIR` directory and that any resource managers are also installed on the system.

i Note

In the following procedures, replace the environment variables with the actual TUXEDO paths as follows: replace `$TUXDIR` with your actual root directory path, and replace `$SYBASE` with the path to the DTM XA Interface installation directory.

The table below provides the SAP-specific information you need to perform the TUXEDO integration. The *TUXEDO Installation Guide* discusses this information in *Integrating a Resource Manager With System/T*.

Type of Information	SAP Specific	Description
RM name	SYBASE_XA_SERVER	The name of the resource manager in the <code><name></code> element of the <code>xa_switch_t</code> structure.
XA structure name	sybase_TUX_xa_switch	The name of the <code>xa_switch_t</code> structure that contains the resource manager identifier, the flags for the resource manager's capabilities, and the function pointers of the XA functions.
Library name	The library files <code>ct_r</code> , <code>cs_r</code> , <code>comn_r</code> , <code>tcl_r</code> , and <code>intl_r</code> which are located in <code>\$SYBASE/\$SYBASE_OCS/lib</code>	The list of files needed to support the DTM XA Interface, and a full path name.
Open string contents	See "Open string parameters for DTM XA Interface" in this document for more information..	The format of the information string passed to the functions.

Note

Whether you should use the DTM XA with the reentrant libraries (such as `ct_r`, `cs_r`, `comn_r`, `tcl_r`, and `intl_r`) depends on whether or not you are developing threaded applications. For threaded applications, you must use reentrant libraries. For non-threaded applications, you can use non-reentrant libraries such as `ct`, `cs`, `comn`, `tcl`, and `intl`.

Related Information

[XA Configuration File for DTM XA Interface \[page 16\]](#)

2.6.1 Linking

The TUXEDO RM file provides information used by TUXEDO utilities to link TUXEDO servers.

Procedure

1. Use the text editor of your choice to open the `$TUXDIR/udataobj/RM` file for editing.
2. Update the file with XA information by adding/verifying entries for SAP resource managers. For most SAP applications, including the `simprpc.ct` sample application, one entry for `SYBASE_XA_SERVER` is all that you need. If you are going to build and run the `rentapp` sample, you may want to go ahead and add the second entry for `SCRAP_XA_SERVER`, as required for `rentapp`.

3. Replace `$$SYBASE/$$SYBASE_OCS` with the fully qualified path to the SAP installation directory containing the XA Interface:

```
SYBASE_XA_SERVER:sybase_TUX_xa_switch:-t -Bstatic -L$$SYBASE/$$SYBASE_OCS/lib  
-lcobct -lxadtm -lct_r -lcs_r -lcomn_r -ltcl_r -lintl_r -Bdynamic -ldl
```

```
SCRAP_XA_SERVER:sybase_TUX_xa_switch:-t -Bstatic -L$$SYBASE/$$SYBASE_OCS/lib  
-lcobct -lxadtm -lct_r -lcs_r -lcomn_r -ltcl_r -lintl_r -Bdynamic -ldl
```

i Note

Each entry *must* be a single continuous line.

The `cobct` libraries are only needed if you are building ESQL/COBOL application servers. If you are not using ESQL/COBOL, you can remove the `-lcobct` specification.

If you want your TUXEDO servers to load and execute all SAP libraries dynamically, you can use entries like the following. Dynamic libraries may increase CPU overhead for TUXEDO server execution.

```
SYBASE_XA_SERVER:sybase_TUX_xa_switch:-L$$SYBASE/$$SYBASE_OCS/lib -lxadtm  
-lct_r -lcobct -lcs_r -lcomn_r -ltcl_r -lintl_r
```

```
SCRAP_XA_SERVER:sybase_TUX_xa_switch:-L$$SYBASE/$$SYBASE_OCS/lib -lxadtm  
-lct_r -lcobct -lcs_r -lcomn_r -ltcl_r -lintl_r
```

You can add a comment line by identifying it with a leading pound sign (#) character.

2.6.2 Setting up the UBBCONFIG File

Examples are provided for setting up the TUXEDO `UBBCONFIG` file with the XA Interface.

Context

For the `simprpc.ct` sample application, the `pubs2` database must be installed on SAP ASE. Use the installation script in the SAP ASE directory under `scripts/installpubs2`.

Procedure

1. Use the ASCII text editor of your choice to open `$$SYBASE/$$SYBASE_OCS/sample/xa-dtm/tuxedo/simprpc.ct/ubbsimpct` for editing. The file is shown here with line numbers to facilitate the discussion:

```
1 *RESOURCES  
2 IPCKEY          123456  
3  
4 MASTER         sybsite  
5 MAXACCESSERS   5
```

```

6  MAXSERVERS      5
7  MAXSERVICES    10
8  MODEL          SHM
9
10 MAXGTT         5
11
12 *MACHINES
13 yourmachine     LMID=sybsite
14 TUXDIR="$TUXDIR"
15 APPDIR="$SYBASE/$SYBASE_OCS/sample/xa-dtm/tuxedo/simprpc.ct"
16 TLOGDEVICE="$SYBASE/$SYBASE_OCS/sample/xadtm/tuxedo/
  simprpc.ct/tuxlog"
17 TLOGNAME=TLOG
18 TUXCONFIG="$SYBASE/$SYBASE_OCS/sample/xa-dtm/tuxedo
  /simprpc.ct/tuxconfig"
19 ULOGPFX="$SYBASE/$SYBASE_OCS/sample/xa-dtm/tuxedo/simprpc.ct/ULOG"
20
21 *GROUPS
22 DEFAULT:        TMSNAME=simprpccttms TMSCOUNT=2
23
24 GROUP1          LMID=sybsite     GRPNO=1
25 OPENINFO="$SYBASE_XA_SERVER: -Uuserid1 -Ppassword1 -Nconnection1"
26
27 *SERVERS
28 simpsrv         SRVGRP=GROUP1    SRVID=1
29
30 *SERVICES

```

2. Replace entries in the file with entries appropriate for your environment as shown in the table below.

Line Number	Entry	Replace with
13	<yourmachine>	Replace with the name of the machine that contains the XA Interface installation. Remember that the machine name is case-sensitive.
14	<\$TUXDIR>	Replace with the actual TUXEDO root directory path.
15, 16, 18, 19	<\$SYBASE/\$SYBASE_OCS>	Replace with the XA Interface installation directory.
22	simprpccttms	This parameter is specific to the <code>simprpc.ct</code> example. In general, this parameter should relate to the value specified in the <code>-o</code> parameter of the <code>buildtms</code> command described on <i>Building the TMS</i> .
25	Open string parameters	See <i>Open String Parameters for the DTM XA Interface</i> for more information.

Results

i Note

See the *TUXEDO Installation Guide* for a detailed discussion of the `UBBCONFIG` file.

Related Information

[Building the TMS \[page 31\]](#)

[Open String Parameters for the DTM XA Interface \[page 13\]](#)

2.6.3 Creating the TUXEDO Configuration File

Create a configuration file from the `UBBCONFIG` file.

Procedure

1. Set the `<$TUXCONFIG>` environment variable to a value that matches the entry in `ubbsimpct` by issuing this command:

```
setenv TUXCONFIG $SYBASE/$SYBASE_OCS/sample/xa-dtm/tuxedo/simprpc.ct/tuxconfig
```

2. Create a TUXEDO configuration file from the `UBBCONFIG` file by executing this command:

```
$TUXDIR/bin/tmloadcf -y <ubconfig_file_name>
```

For this verification, using the `simprpc.ct` sample, replace `<ubconfig_file_name >` with `ubbsimpct`.

2.6.4 Building the TMS

Build the transaction monitor server (TMS) by executing the `<SYBASE_XA_SERVER>` command, where `<output_file>` is a name you choose for the transaction monitor server program.

```
$TUXDIR/bin/buildtms -r <SYBASE_XA_SERVER> -o  
$TUXDIR/bin/<output_file>tms
```

It is helpful to append `tms` to the name as shown here, so it is easily identified. Choose a unique name for the program so that it does not conflict with TMS programs for other resource managers (`TMS`, `TMS_D`, and `TMS_SQL` are reserved).

For the `simprpc.ct` example verification, the `UBBCONFIG` file uses `simprpccttms`. See *Setting up the UBBCONFIG File*.

The program is stored in `$TUXDIR/bin` so that the TUXEDO System/T start program can find it.

2.6.5 Building the COBOL Runtime Environment

Configuring CICS to Support SAP XA COBOL Transactions. In CICS transactions, COBOL transactions use the COBOL runtime, which you must modify to communicate with the SAP XA environment.

Prerequisites

You must use MicroFocus COBOL 3.1 or higher.

Procedure

1. Log in as root.
2. Set the `<COBDIR>` environment variable to the directory path for the MicroFocus COBOL installation.
3. Set the `<PATH>` environment variable to include the MicroFocus COBOL binary directory.
4. Change directory to the `$SYBASE/$SYBASE_OCS/sample/xa-dtm/cics` directories.
5. Run `xa_make_cobol_runtime`.

Results

Caution

This script assumes that the CICS COBOL runtime file is installed in `/usr/lpp/cics/v1.1/bin`. If you have installed CICS somewhere else, you must edit this script to reflect your installation.

This script builds a MicroFocus COBOL runtime environment with CICS and SAP XA support. It allows CICS transactions written in COBOL to reference XA Interface and Open Client functions at run time. The script takes several minutes to run. For more information, see your CICS documentation.

3 Application Programming Guidelines

Embedded SQL and Client-Library applications must conform to certain coding constraints in order to function within the SAP XA environment.

3.1 X/Open DTP Versus Traditional SAP Transaction Processing

The X/Open DTP model of transaction processing differs substantially from the traditional SAP model. The traditional SAP TP environment is connection oriented. Programs set up connections directly between the application program and SAP ASE using connection management SQL statements.

In the XA Interface environment, the XA Interface, using LRMs, sets up connections for the application.

Traditional TP model	X/Open DTP model
There is one or more transaction per client/server connection.	There is no notion of connections. Components communicate through interfaces.
Transactions are usually local, with each transaction confined to a single SAP ASE server.	Transactions are global. They span resource managers. The work done within a transaction is accomplished using more than one resource manager.
Each SAP ASE server is responsible for the recovery of the data it contains.	The transaction manager is responsible for recovering the data stored in all of the resource managers.

3.2 Managing Transactions and Connections

Applications must pay special attention to commands related to managing transaction and connections, including the current connection.

i Note

The XA Interface uses an ANSI default isolation level of 3. To minimize read-only locking, programs can set the transaction isolation level in the XA configuration file, or they can use `select xxx from table noholdlock` in individual SQL operations. See the *Transact-SQL Users Guide* for additional information on transaction isolation levels.

3.2.1 Managing Transactions

The CICS, Encina, or TUXEDO TM is responsible for transaction management. This includes creating a global transaction in which all of an application's work is either committed or rolled back.

Consequently, applications cannot issue SQL statements that manage transactions.

Specifically, applications cannot invoke the following Embedded SQL commands:

- `begin transaction`
- `commit`
- `rollback`

Client-Library applications cannot execute (via `ct_command`, `ct_dynamic`, or `ct_cursor`) any of these Transact-SQL commands:

- `begin transaction`
- `commit transaction`
- `rollback transaction`
- `set (chained, noexec, isolation, parseonly, statistics io, statistics time)`
- `save transaction`

i Note

The application must recognize SAP ASE-detected errors, and abort or roll back the transaction through the TM. This is especially important for SAP ASE detected-deadlocks.

3.2.2 Managing Connections

Applications rely on the SAP XA environment for management of client/server connections.

Connection management occurs transparently to the application. Consequently, Embedded SQL applications cannot invoke the following commands for XA-managed connections:

- `connect`
- `disconnect`

Client-Library applications cannot call these Client-Library commands using XA-managed connections:

- `ct_close`
- `ct_con_alloc`
- `ct_con_drop`
- `ct_con_props`
- `ct_config` with the parameters:
 - `CS_ENDPOINT`
 - `CS_EXPOSE_FMTS`
 - `CS_HIDDENKEYS`
 - `CS_MAX_CONNECT`

- CS_NETIO
- CS_TRANSACTION_NAME
- ct_connect
- ct_exit
- ct_getloginfo
- ct_init
- ct_options with the parameters:
 - CS_OPT_CHAINXACTS
 - CS_OPT_FORCEPLAN
 - CS_OPT_FORMATONLY
 - CS_OPT_NOEXEC
 - CS_OPT_PARSEONLY
 - CS_OPT_STATS_IO
- ct_remote_pwd
- ct_setloginfo
- CS_OPT_STATS_TIME

In addition, Client-Library applications cannot call these CS-Library commands:

- cs_ctx_drop (with global context handle)
- cs_objects (CS_CLEAR, CS_SET)

3.2.3 The Current Connection

The notion of a default connection, as described in the Open Client Embedded SQL documentation, does not exist in the SAP XA environment.

Consequently, applications must always explicitly specify a current connection.

There are two ways to specify the current connection in Embedded SQL. They are:

- The `set connection` command
- The `at <connection name>` clause

A current connection does not span transactions. For example, an application must reset the current connection after each CICS `SYNCPOINT` command or Encina `onCommit` command. To avoid confusion about the scope of the current connection, SAP recommends that you use the `at <connection_name>` clause with all Embedded SQL statements.

3.2.4 Nontransactional Connections

Applications can open and use nontransactional connections with the normal Open Client or Embedded SQL Interfaces.

Operations on such connections do not participate in the transaction and are not committed or rolled back. They may be useful for queries of unchanging databases and updates of data which can be inaccurate.

3.3 Deallocating Cursor Function with Client-Library

Application programs use and reuse connections that have been allocated for them via the XA Interface. The SAP implementation of cursors starting with SQL Server version 10.1 requires cursor structures on both the client (TM/RM program) side and the SAP ASE side.

When a client explicitly deallocates a cursor, or when the client connection is closed, SAP ASE deallocates the server cursor structures.

When the first iteration of a program opens or closes a cursor but the connection stays allocated (as it does with XA-Library), the second iteration of the same program fails, as it attempts to open the same cursor name. SAP ASE informs us that it already has a cursor by this name at the same nesting level.

The application program must explicitly close and deallocate the cursor before it commits or aborts its transaction. This must be done in the transaction program that allocates the cursor. Embedded SQL records information about cursors which allows the XA Interface to perform the deallocation.

With Client-Library, you must handle error paths so that cursors are deallocated before a TM abort code is called. That is, if the open cursor works, deallocate it.

Use `ct_cursor()` with type `CS_CURSOR_CLOSE` and option `CS_DEALLOC`.

3.4 Dynamic SQL

The use of dynamic SQL statements has many characteristics in common with cursors, with the additional complexity that temporary stored procedures are sometimes placed into SAP ASE.

The use of dynamic SQL is not recommended in transactional applications, but if they are used, the following guidelines must be adhered to:

- In Embedded SQL use “Method 3: Prepare and Fetch with a Cursor” (see the ESQL document or a description of this method) if possible. When this method is used, Embedded SQL places information in the system which allows the XA Interface to locate and deallocate all dynamic SQL and cursors.
- In all other cases, the dynamic SQL statements and all associated cursors must be closed and deallocated to avoid adverse effects on other transactions. Any associated Client-Library command structures should be dropped to avoid memory leaks. See the Open Client and ESQL documentation for information on how to drop these command structures.

3.5 Obtaining a Client-Library Connection Handle

Obtaining a connection handle is an issue specific to Client-Library applications.

When the TM opens a connection to SAP ASE, the XA Interface allocates a `CS_CONNECTION` structure for its own use. Once control passes to the application, that application must use the connection handle contained in this structure.

To get the connection handle, specify `CS_GET` for the `cs_object` routine's `action` parameter with an object type of `CS_CONNECTION`. `cs_object`'s `objdata` parameter returns a structure containing a `connection` field. This field contains the handle to the `CS_CONNECTION` structure.

⚠ Caution

The XA Interface also allocates a `CS_COMMAND` structure whose handle is returned in the `command` field of the structure to which the `objdata` parameter points. An application cannot use this command handle, as the XA Interface continues to use this handle, itself.

The following code fragment demonstrates how to retrieve the handle to the `CS_CONNECTION` structure:

```
/*
** Arguments:
** connection null-terminated name of the connection
** (ESQL) or LRM connH loaded with the CS_CONNECTION
** handle if the lookup is successful
**
** Returns:
** CS_SUCCEED connection handle found successfully
** CS_FAIL unable to find connection handle for given
** connection /#include <stdio.h> #include <strings.h>
** #include <cspublic.h>CS_RETCODE getConn(connection,
** connH)CS_CHAR connection[128];CS_CONNECTION connH;
{
CS_INT retcode;
CS_CONTEXT *ctx;
CS_OBJNAME name;
CS_OBJDATA data;
CS_THREAD thread_functions;
CS_INT outlen;
#define THREADID_SIZE 8
CS_BYTE thread_id[THREADID_SIZE];
/* Check arguments */
if (strlen(connection) >= 128)
{
/* Connection name is too long */
return(CS_FAIL);
}
/* Get the global context handle */
retcode = cs_ctx_global(CS_VERSION_100, &ctx);
if (retcode != CS_SUCCEED)
{
/* Major environment problems! */
return(CS_FAIL)
}
/*
** Initialize the CS_OBJNAME structure to look
** for the specified connection name.
*/
name.thinkexists = CS_FALSE;
name.object_type = CS_CONNECTNAME;
strcpy(name.last_name, connection);
name.fnlen = CS_UNUSED;
name.lnlen = CS_NULLTERM;
name.scopelen = CS_UNUSED;
/*
** Set the current thread-id so we get the instance of
** this connection that this thread should be using.
*/
retcode = cs_config(ctx, CS_GET,
CS_THREAD_RESOURCE, &thread_functions,
CS_UNUSED, &outlen);
if (retcode != CS_SUCCEED)
{
```

```

/*
** Even in an non-threaded environment, this should be
** successful.
*/
return(CS_FAIL);
}
name.thread = (CS_VOID *) thread_id;
retcode = (*thread_functions.thread_id_fn)(
name.thread, THREADID_SIZE,
&name.threadlen);
if (retcode != CS_SUCCEED)
{
return(CS_FAIL);
}
/*
** Initialize the CS_OBJDATA structure to receive the
** connection handle for this connection name
*/
data.actuallyexists = CS_FALSE;
data.connection = (CS_CONNECTION *) NULL;
data.command = (CS_COMMAND *) NULL;
data.buffer = (CS_VOID *) NULL;
data.bufllen = CS_UNUSED;
/* Retrieve the connection information */
retcode = cs_objects(ctx, CS_GET, &name,
&data);
if (retcode == CS_SUCCEED)
{
if (data.actuallyexists == CS_TRUE)
{
*connH = data.connection;
return(CS_SUCCEED);
}
else
{
/* No connection by that name exists */
return(CS_FAIL);
}
}
else
{
/*
** The global CS_CONTEXT handle is probably not
** initialized with connection information yet
*/
return(CS_FAIL);
}
}
}

```

3.6 Multiple-thread Environment Issues

Threads are multiple, simultaneous paths of execution in a single operating system process, and share access to the resources allocated to that process.

Some application programming interfaces (APIs) allow an application developer to effectively use threads in the transaction environment. In turn, The SAP XA Interface supports a maximum level of concurrency, enabling it to take advantage of those environments.

However, this raises several issues for an application developer. For background information and a complete discussion of the issues, see the OSF's *DCE Application Developer's Guide*.

The *Open Client Reference Manual* contains a section on thread-safe programming. XA Interface assigns connections to threads at the request of the TM. These assignments ensure that only one thread at a time is working on the connection and is the reason the thread ID is included in the `cs_object` request described in *Obtaining a Client-Library Connection Handle*. As long as connections assigned by XA Interface are used in the thread to which they are assigned and the restrictions on their use are followed, there should be no Open Client or ESQL threading-related problems.

Related Information

[Obtaining a Client-Library Connection Handle \[page 36\]](#)

3.6.1 Caveats of Thread Use

Client-Library uses a connection state machine to verify that applications call Client-Library routines in a logical sequence.

See, Program Structure in the *Open Client Client-Library/C Programmer's Guide* for an explanation of the steps involved in structuring a Client-Library application.

The assumption underlying the use of threads is that when a thread disassociates from a transaction branch, it leaves the state machine in an inactive state. By default, all Embedded SQL statements leave the connection quiescent. With Client-Library, this is true only in the following circumstances:

- When `ct_results` returns `CS_END_RESULTS`, or `CS_SUCCEED` with a result type of `CS_CURSOR_RESULT`.
- After an application calls `ct_cancel` with `<type>` as `CS_CANCEL_ALL`.
- When an application returns `CS_CANCELED`. The APIs that return `CS_CANCELED` include `ct_send()`, `ct_results()`, and `ct_get_data()`.

Caution

If connections are not left in an inactive state, the consequences may include transaction rollbacks, extra overhead as the XA Interface cleans up the connection (which may require full connection close and reopen), and the possible failure of subsequent transactions. In such a situation, XA Interface attempts to maintain application operation while it minimizes failure.

3.6.2 Embedded SQL Thread-Safe Code

Thread-safe code protects the use of shared resources with a mutex (MUTual EXclusion semaphore). A mutex protects shared resources, such as files and global variables, by preventing them from being accessed by more than one thread at a time.

Use the `-h` (UNIX) or `/threadsafe` (VMS) precompiler option to generate thread-safe code.

3.6.3 Tightly Coupled Transactions

The XA environment treats each thread or process that works on a transaction as a transaction branch. Each transaction branch is assigned a different `xid` and works independently of the other branches.

However, all branches are committed or rolled back as a unit. This applies to MTS/COM+ environments only, and only if you are using ctlib-based drivers that are older than version 3.6.

Some TMs allow branches to be tightly coupled. Tightly coupled branches are assigned the same `xid` and work together on the transaction. In such cases the open string can contain the `-01` option. This option causes SAP ASE to move the work among connections on demand and eliminates any lock that might otherwise occur between the connections. See your TM documentation to determine how the TM can be configured for tightly coupled operation.

⚠ Caution

Set the `-01` option only when the application design is guaranteed to avoid conflicting updates. Normally this is true only when the application branches are fully serialized, (branch B operates only after branch A completes). Data inconsistency may occur if the interaction of the tightly coupled branches is not well designed.

Without the `-01` option, attempts by the branches to update the same database row can result in a deadlock internal to the transaction. The `-01` option has no practical effect when the branches are not tightly coupled through the TM and are assigned different `xids`.

⚠ Caution

Cursors and dynamic SQL cannot be retained when the transaction is assigned to a different connection. Therefore, they should not be used unless the application structure guarantees that they are opened and closed during a period when no other branch will work on the transaction.

i Note

The transaction is reassigned to another connection only between batches. A tightly coupled application can ensure that a set of operations is completed without conflict by performing all the operations in a single batch. This implies that operations within a single stored procedure are also completed without conflict.

For example, if row z in table B must contain the sum of rows x and y in table A. The following can result in an invalid value in row z:

```
Branch 1:                               Branch 2
  Updates Row x -> 5
  Reads Row y (= 4)
                                           Updates Row y -> 5
                                           Reads Row x (= 5)
                                           Updates Row z -> 10
  Updates Row z -> 9 (wrong value)
```

No problem occurs if the branches are performed serially:

```
Branch 1:                               Branch 2
  Updates Row x -> 5
  Reads Row y (= 4)
  Updates Row z -> 9
                                           Updates Row y -> 5
```



```
Reads Row x (=5)
Updates Row z -> 10
```

A control branch can also be used to resolve the problem:

```
Branch 0:          Branch 1:          Branch 2
(controller)
Starts Branches 1 and 2
Waits for both to complete
                Updates Row x      Updates Row y -> 5
                Terminates        Terminates
    Reads Row y
    Reads Row x
    Updates Row z
```

TM specific branch control mechanisms must be used to implement these serialization mechanisms.

3.7 Linking with CT Library

The XA Interface requires that the application be linked with the threaded versions of the Open Client Libraries.

See the *Open Client/Server Supplement* for your platform to identify the libraries you must specify. If you do not link the proper thread-safe libraries, you may experience a variety of Open Client failures.

3.8 Sample Embedded SQL COBOL Fragment

Example of code fragment that sets the current connection, and inserts data into an SAP ASE database.

```
*REMARKS. TRANSACTION-ID IS 'POPS'.
*THIS TRANSACTION POPULATES A DATABASE'S DATA TABLE *WITH STOCK DATA ENTRIES.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY DFHBMSCA.
COPY DFHAID.
COPY AIXCSET.
EXEC SQL INCLUDE SQLCA END-EXEC.
77 RESPONSE          PIC 9(8) COMP.
01 MSG-LIST.
02 MSG-1              PIC X(70) VALUE
'Transaction Failed: Unable To Prime Stock'
-'Table.'.
02 MSG-2              PIC X(70) VALUE
'Stock Records Added Successfully.'.
01 TRANSFAIL         PIC X(70).
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STOCK-RECORD.
02 STOCK-NUM         PIC X(5).
02 ITEM-DESC         PIC X(30).
02 STOCK-QTY         PIC X(7).
02 UNIT-PRICE PIC S9(4)V99 VALUE ZEROES.
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
* CHECK BASIC REQUEST TYPE
```

```

*
IF EIBAID = DFHCLEAR
EXEC CICS SEND CONTROL FREEKB
END-EXEC
EXEC CICS RETURN
END-EXEC
END-IF.
* MAIN PROCESSING
*SET UP STOCK RECORD DETAILS AND THEN WRITE OUT
*STOCK RECORD.
*
MOVE '31421'TO STOCK-NUM.
MOVE 'Widget (No.7)'TO ITEM-DESC.
MOVE '0050035'TO STOCK-QTY.
MOVE 25.55 TO UNIT-PRICE.
PERFORM WRITE-STOCKREC.
MOVE '43567'TO STOCK-NUM.
MOVE 'Splunkett ZR-1'TO ITEM-DESC.
MOVE '0005782'TO STOCK-QTY.
MOVE 143.79 TO UNIT-PRICE.
PERFORM WRITE-STOCKREC.
EXEC CICS SYNCPOINT
RESP (RESPONSE)
END-EXEC.
IF RESPONSE NOT = DFHRESP (NORMAL)
MOVE MSG-1 TO TRANSFAIL
PERFORM FAIL-TRANS
END-IF.
MOVE MSG-2 TO MSGOUTO.
EXEC CICS SEND MAP ('MSGLINE')
MAPSET ('AIXCSET')
FREEKB
END-EXEC.
EXEC CICS RETURN
END-EXEC.
GOBACK.
* ATTEMPT TO WRITE OUT NEW STOCK RECORD.
*
WRITE-STOCKREC.
EXEC SQL SET CONNECTION connection_2
END-EXEC
IF SQLCODE NOT = 0
MOVE MSG-1 TO TRANSFAIL
PERFORM FAIL-TRANS
END-IF.
EXEC SQL INSERT INTO STOCK VALUES (:STOCK-RECORD)
END-EXEC
IF SQLCODE NOT = 0
MOVE MSG-1 TO TRANSFAIL
PERFORM FAIL-TRANS
END-IF.
* IF UNABLE TO APPLY CREATE, END TRANSACTION
* AND DISPLAY REASON FOR FAILURE.
*
FAIL-TRANS.
MOVE TRANSFAIL TO MSGOUTO
EXEC CICS SEND MAP ('MSGLINE')
MAPSET ('AIXCSET')
FREEKB
END-EXEC
EXEC CICS RETURN
END-EXEC.

```

3.9 Sample Embedded SQL C Fragment

Example of how to set the current connection, and accesses data stored on SAP ASE.

```
EXEC SQL INCLUDE sqlca;
int rcode;
EXEC SQL BEGIN DECLARE SECTION;
char name[15];
char supplier[30];
char supplier_address[30];
int order_quantity;
EXEC SQL END DECLARE SECTION;
main()
{
char errmsg[400];
char qmsg[400];
short mlen;
EXEC SQL WHENEVER SQLERROR GOTO :errexit;
EXEC SQL WHENEVER SQLWARNING GOTO :errexit
EXEC SQL WHENEVER NOT FOUND GOTO :errexit
/* Get addressability for EIB... */
/*
** Write record to CICS temporary storage queue...
*/
/* Send the first map */
EXEC CICS SEND MAP("PANEL1") MAPSET("UXA1")
FREEKB ERASE RESP(rcode);
if (rcode != DFHRESP(NORMAL))
EXEC CICS ABEND ABCODE("X001");
/* Receive the response */
EXEC CICS RECEIVE MAP("PANEL1") MAPSET("UXA1")
RESP(rcode);
if (rcode != DFHRESP(NORMAL))
EXEC CICS ABEND ABCODE("X002");
/* Select a record from the table based on user input. */
sprintf(name, "%s", panel1.panelli.newnamei);
EXEC SQL SET CONNECTION connection_1;
EXEC SQL SELECT name, supplier, supplier_address, order_quantity
INTO
:name, :supplier, :supplier_address, :order_quantity
FROM cheese
WHERE name = :name;
/* Handle "no rows returned" from SELECT */
if (sqlca.sqlcode == 100)
{
sprintf(panel4.panel4o.messageo, "%s",
NOCHEESE);
EXEC CICS SEND MAP("PANEL4") MAPSET("UXA1") FREEKB ERASE RESP(rcode);
if (rcode != DFHRESP(NORMAL))
EXEC CICS ABEND ABCODE("X009");
EXEC CICS SEND CONTROL FREEKB;
EXEC CICS RETURN;
}
/* Fill in and send the second map */
memset (&panel2.panel2o, '0',
sizeof(panel2.panel2o));
sprintf(panel2.panel2o.nameo, "%s", name);
sprintf(panel2.panel2o.supplo, "%s", supplier);
sprintf(panel2.panel2o.addresso, "%s",
supplier_address);
sprintf(panel2.panel2o.ordero, "%d", order_quantity);
EXEC CICS SEND MAP("PANEL2") MAPSET("UXA1")
FREEKB ERASE RESP(rcode);
if (rcode != DFHRESP(NORMAL))
EXEC CICS ABEND ABCODE("X003");
```

```

/* Receive the response */
if (panel2.panel2i.questi == 'y')
{
/* Send the third map... */
/* Receive the response... */
/* Update the database */
order_quantity = atoi(panel3.panel3i.newordi);
EXEC SQL UPDATE cheese
set order_quantity = :order_quantity
where name = :name;
/* Write a record to the temporary queue */
sprintf(qmsg, "%s", "The cheese table was updated");
mlen = strlen(qmsg);
EXEC CICS WRITEQ TS QUEUE("TEMPXAQ1")
FROM(qmsg) LENGTH(mlen) RESP(rcode);
if (rcode != DFHRESP(NORMAL))
EXEC CICS ABEND ABCODE("X010");
}
else
{
/*
** The user does not wish to update so free the keyboard and return...
*/
}
/* Commit the update */
EXEC CICS SYNCPOINT RESP(rcode);
if (rcode != DFHRESP(NORMAL))
EXEC CICS ABEND ABCODE("X011");
/*
** Send the fourth map confirming successful update...
*/
EXEC CICS RETURN;
errexit:
fprintf(stderr, "error in cheeseland %d\n", sqlca.sqlcode);
/* Handle general errors */
sprintf(errmsg, "%.60s\n", sqlca.sqlerrm.sqlerrmc);
strncpy(panel4.panel4o.messageo, errmsg, 60);
sprintf(panel4.panel4o.codeo, "%d", sqlca.sqlcode);
/*
** Send the fourth map with appropriate message...
*/
/* Rollback the transaction */
EXEC CICS SYNCPOINT ROLLBACK;
EXEC CICS SEND CONTROL FREEKB;
EXEC CICS RETURN;
}



```

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
 - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
 - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

© 2021 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.