Using Adaptive Server
Distributed Transaction Management Features

Adaptive Server Enterprise

Version 12

Document ID: 31650-01-1200-02

Last revised: October 1999

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Contents

About This Book

Audience

This manual is intended for administrators or application developers who have purchased the Adaptive Server Distributed Transaction Management (DTM) feature.

How to use this book

Read this manual after you have installed Adaptive Server and its associated feature licenses.

Related documents

If you are using the DTM feature in an X/Open XA environment, also read the *XA Interface Integration Guide for CICS, Encina, and TUXEDO*.

Other sources of information

Use the Sybase Technical Library CD and the Technical Library Web site to learn more about your product:

- Technical Library CD contains product manuals and technical documents and is included with your software. The DynaText browser (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting Technical Library.

- Technical Library Web site includes the Product Manuals site, which is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition, you'll find links to the Technical Documents Web site (formerly known as Tech Info Library), the Solved Cases page, and Sybase/Powersoft newsgroups.

  To access the Technical Library Web site, go to the Sybase Support Services site at support.sybase.com, click the Electronic Support Services tab, and select a link under the Technical Library heading.

Sybase certifications on the web

Technical documentation at the Sybase Web site is updated frequently.

❖ **For the latest information on product certifications and/or the EBF Rollups:**

1. Point your Web browser to Technical Documents at http://www.techinfo.sybase.com.
2. In the Browse section, click on the What's Hot entry.

3 Explore your area of interest: Hot Docs covering various topics, or Hot Links to Technical News, Certification Reports, Partner Certifications, and so on.

❖ **If you are a registered SupportPlus user:**

1 Point your Web browser to Technical Documents at http://www.techinfo.sybase.com.

2 In the Browse section, click on the What's Hot entry.

3 Click on the EBF Rollups entry.

You can research EBFs using Technical Documents, and you can download EBFs using Electronic Software Distribution (ESD).

4 Follow the instructions associated with the SupportPlus^{SM} Online Services entries.

❖ **If you are not a registered SupportPlus user, and you want to become one:**

You can register by following the instructions on the Web.

To use SupportPlus, you need:

1 A Web browser that supports the Secure Sockets Layer (SSL), such as Netscape Navigator 1.2 or later

2 An active support license

3 A named technical support contact

4 Your user ID and password

❖ **Whether or not you are a registered SupportPlus user:**

You may use Sybase's Technical Documents. Certification Reports are among the features documented at this site.

1 Point your Web browser to Technical Documents at http://www.techinfo.sybase.com

2 In the Browse section, click on the What's Hot entry.

3 Click on the topic that interests you.

Conventions

The following style conventions are used in this manual:

- In a sample screen display, commands you should enter exactly as shown are given in:

```
this font
```

- In a sample screen display, words which you should replace with the appropriate value for your installation are shown in:

  this font

- In the regular text of this document, the names of files and directories appear in italics:

  /usr/u/sybase

- The names of programs, utilities, procedures, and commands appear in bold type:

  bcp

- Commands for both the C and Bourne shells are provided in this document when they differ. The initialization file for the C shell is called *.cshrc*. The initialization file for the Bourne shell is called *.profile*. If you are using a different shell, such as the Korn shell, refer to your shell-specific documentation for the correct command syntax.

The conventions for syntax statements in this manual are as follows:

Table 1: SQL syntax conventions

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords are in bold.
variable	Variables, or words that stand for values that you fill in, are in *italics*.
{ }	Curly braces indicate that you choose at least one of the enclosed options. Do not include braces in your option.
[ ]	Brackets mean choosing one or more of the enclosed options is optional. Do not include brackets in your option.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you can select only one of the options shown.
,	The comma means you can choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Overview

Adaptive Server version 12 introduces several distributed transaction management features to:

- Bring Adaptive Server into full compliance with the X/Open XA protocol when acting as a resource manager, without requiring additional services such as XA-Server
- Provide support for distributed transactions coordinated by Microsoft Distributed Transaction Coordinator (MSDTC)
- Ensure consistent commit or rollback for transactions that update Adaptive Server version 12 data via remote procedure calls (RPCs) and Component Integration Services (CIS)
- Provide the framework to support additional distributed transaction management protocols in the future

This chapter presents an overview of new distributed transaction management features, and describes changes to Adaptive Server that support those features.

Distributed Transaction Management Features

Adaptive Server version 12 introduces these distributed transaction management features:

- Improved transaction and thread management. Adaptive Server version 12 manages all transactions as server resources, and provides the ability to attach and detach threads from transactions. These new capabilities provide a common interface for supporting clients of local server transactions, as well as clients in X/Open XA and MSDTC environments. See “Configuring Transaction Resources” on page 10.
- New distributed transaction coordination services. Adaptive Server provides consistent rollback and commit capabilities for transactions that modify data in remote Adaptive Servers via RPCs and CIS. New transaction coordination services guarantee the integrity of such distributed transactions, even when no external transaction manager is present. See Chapter 3, “Using Adaptive Server Transaction Coordination Services”.
- Improved recovery for prepared transactions. During recovery, Adaptive Server identifies prepared transactions that were coordinated by the X/Open XA protocol and Adaptive Server native transaction coordination services. Adaptive Server restores these transactions to the condition they were in prior to recovery, and brings the associated database online more quickly than in previous server versions. See “Crash Recovery Procedures for Distributed Transactions” on page 32.
- New **dbcc** commands for heuristically completing distributed transactions. See “Heuristically Completing Transactions” on page 34.

Affected Transaction Types

The new Adaptive Server version 12 DTM features affect:

- Distributed transactions coordinated by external transaction managers
- Transactions that update data using RPCs and CIS

Distributed Transactions Coordinated by External Transaction Managers

Distributed transactions can take place in an environment where an external transaction manager coordinates transaction execution using a specific protocol, such as X/Open XA. Adaptive Server supports transactions using the CICS, Encina, TUXEDO, and MSDTC transaction managers through the DTM XA interface to Adaptive Server.

Note Adaptive Server version 12 with the DTM XA interface provides features that were previously part of the XA-Server product. The XA-Server product is not required and is not included with Adaptive Server version 12. See the *XA Interface Integration Guide for CICS, Encina, and TOPEND* for information about the DTM XA interface.

New Behavior for Transaction Manager-Coordinated Transactions

Adaptive Server version 12 natively implements several features that were part of the XA-Library and XA-Server products, and provides new recovery procedures for prepared transactions coordinated via the X/Open XA protocol. See “Configuring Transaction Resources” on page 10 and “Crash Recovery Procedures for Distributed Transactions” on page 32 for more information.

The XA interface to Adaptive Server has been modified to accommodate the server’s new distributed transaction management features. Changes to the XA interface are transparent to X/Open XA client applications. However, you must link the version 12 DTM XA interface with your X/Open XA transaction manager in order to use Adaptive Server version 12 as a resource manager. Details on all XA interface changes are described in the *XA Interface Integration Guide for CICS, Encina, and TOPEND*.

Adaptive Server version 12 also introduces support for distributed transactions coordinated by MSDTC. MSDTC clients can communicate directly with Adaptive Server using the native interface. Clients can also communicate with one or more Adaptive Server running on UNIX by using the DTM XA Interface.

Note MSDTC clients using the DTM XA interface must possess *dtm_tm_role* in the Adaptive Server(s) they access. See the *XA Interface Integration Guide for CICS, Encina, and TUXEDO* for more information about *dtm_tm_role*.

RPC and CIS Transactions

Local Adaptive Server transactions can update data in remote servers by using Transact-SQL remote procedure calls (RPCs) and Component Integration Services (CIS). RPC updates are accomplished by executing an RPC from within a locally-created transaction. For example:

```
sp_addserver westcoastsrv, ASEnterprise, hqsales
begin transaction rpc_tran1
update sales set commission=300 where salesid=”12OZ”
exec westcoastsrv.salesdb..recordsalesproc
commit rpc_tran1
```

The above transaction updates the *sales* table on the local Adaptive Server, but also updates data on a remote server using the RPC, *recordsalesproc*.

CIS provides a way to update data on remote tables as if those tables were local. By using **sp_addobjectdef**, users can create local objects in Adaptive Server that reference remote data. Updating the local object modifies data in the remote Adaptive Server. For example:

```
sp_addobjectdef salesrec,
“westcoastsrv.salesdb..sales”, “table”
begin transaction cis_tran1
update sales set commission=300 where salesid=”12OZ”
update salesrec set commission=300 where salesid=”12OZ”
commit cis_tran1
```

New Behavior for RPC and CIS Transactions

Prior to Adaptive Server version 12, transactions that updated data via RPCs and CIS could not roll back the work of the remote server, nor could those transactions be assured that the remote work actually committed. Adaptive Server provides new transaction coordination services to assure that RPCs and CIS updates commit or roll back their work with the initiating transaction. See Chapter 3, “Using Adaptive Server Transaction Coordination Services” for more details.

If you have applications that rely on the earlier behavior of RPCs and CIS updates, you can disable transaction coordination services. See “enable xact coordination Parameter” on page 9 for information.

SYB2PC Transactions

SYB2PC transactions use the Sybase two-phase commit protocol to ensure that the work of a distributed transaction is committed or rolled back as a logical unit.

Adaptive Server version 12 does not modify the behavior of SYB2PC transactions. However, application developers who implement SYB2PC transactions may want to consider using Adaptive Server transaction coordination services instead. Compared to SYB2PC transactions, transactions coordinated directly by Adaptive Server use fewer network connections and execute more quickly, while still ensuring the integrity of the distributed transaction. Application code can also be simpler when Adaptive Server, rather than the application, coordinates remote transactions. See Chapter 3, “Using Adaptive Server Transaction Coordination Services” for more information.

CHAPTER 2

Enabling DTM Features

This chapter describes how to enable Adaptive Server DTM Features. It includes these sections:

Installing a License Key

Distributed Transaction Management is available as a separately-licensed Adaptive Server feature. Before you can enable and use DTM features, you must purchase and install a valid license for both Adaptive Server and the DTM feature.

See your *Installation Guide* for information about installing license keys and using Sybase Software Asset Management (SySAM). Contact your Sybase sales representative if you want to purchase a license for DTM or other licensed Adaptive Server features.

Enabling DTM Features

After you have purchased and installed a valid license for Adaptive Server and the DTM feature, you can enable DTM features by using **sp_configure** with the **enable dtm** and **enable xact coordination** configuration parameters.

enable dtm Parameter

The **enable dtm** parameter enables or disables basic DTM features. When **enable dtm** is set to 1 (on), Adaptive Server supports external transactions from MSDTC, and from X/Open XA transaction managers via the DTM XA Interface. See the *XA Interface Integration Guide for CICS, Encina, and TUXEDO* for more information.

To enable basic DTM Features, use the command:

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

You must reboot Adaptive Server for this change to take effect.

enable xact coordination Parameter

enable xact coordination enables or disables Adaptive Server transaction coordination services. When this parameter is enabled, Adaptive Server ensures that updates to remote Adaptive Server data commit or roll back with the original transaction. See Chapter 3, "Using Adaptive Server Transaction Coordination Services" for more information.

To enable transaction coordination, use the command:

```
sp_configure 'enable xact coordination', 1
```

You must reboot Adaptive Server for this change to take effect.

Configuring Transaction Resources

Adaptive Server version 12 provides a common interface to support both local server transactions and external transactions coordinated by distributed transaction protocols. Distributed transaction protocol support is provided for X/Open XA, MSDTC, and native Adaptive Server transaction coordination services.

Adaptive Server manages all transactions as configurable server resources, and the System Administrator can configure the total number of resources available in a given server. Client tasks that access Adaptive Server in an X/Open XA environment can also suspend and join threads to transaction resources as needed.

This section describes how to determine and configure the total number of transaction resources available to Adaptive Server.

Calculating Required Transaction Descriptors

Adaptive Server version 12 uses the **transaction descriptor** resource to manage transactions within a server. A transaction descriptor is an internal memory structure that Adaptive Server uses to represent a transaction.

Upon booting up, Adaptive Server allocates a fixed number of transaction descriptors based on the value of the configuration parameter, **txn to pss ratio**, and places them in a pool. Adaptive Server obtains transaction descriptors from the pool as they are needed for new transactions. As transactions complete, descriptors are returned to the pool. If there are no transaction descriptors available, transactions may be delayed as Adaptive Server waits for descriptors to become freed.

To properly configure the number of transaction descriptors, it is important that you understand exactly when Adaptive Server tries to obtain new descriptors from the global pool. A new transaction descriptor is required when:

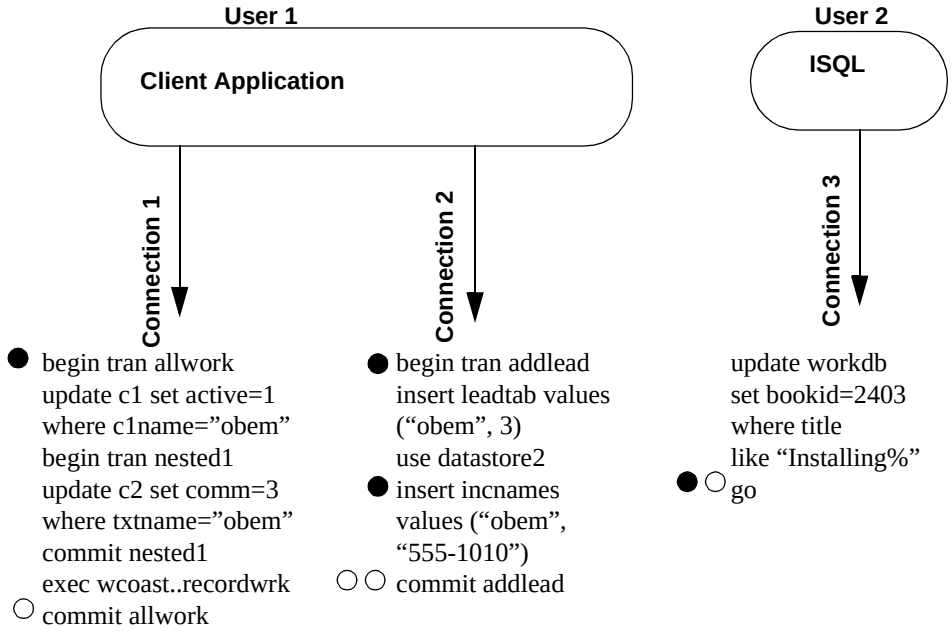
- A client connection initiates a new, outer-level transaction. This can occur explicitly, when the client executes an outer-level **begin transaction** command. It can also occur implicitly, when a client modifies data without entering a **begin transaction** command.

  Once an outer-level transaction has begun, future nested **begin transaction** commands do not require additional transaction descriptors. Allocation and deallocation of the transaction descriptor is dictated by the outer-most block of the transaction.

- An existing transaction modifies a second database (a multi-database transaction). A multi-database transaction requires a dedicated transaction descriptor for *each* database it accesses.

Figure 2-1 illustrates how Adaptive Server obtains and releases transaction descriptors for different transaction types.

Figure 2-1: Allocating and deallocating transaction descriptors



In Figure 2-1, Adaptive Server uses a total of three transaction descriptors for User 1, who accesses the server through a pair of client connections. The server allocates a single descriptor for transaction "allwork," which is freed when that transaction commits. The nested transaction, "nested1," does not require a dedicated transaction descriptor.

Transaction “addlead,” a multi-database transaction, requires two transaction descriptors—one for the outer transaction block, and one for modifying a second database, “datastore2.” Both transaction descriptors are released when the outer transaction block commits.

User 2, accessing Adaptive Server from **isql**, also requires a dedicated transaction descriptor. Even though User 2 did not explicitly create an outer transaction block with **begin transaction**, Adaptive Server implicitly creates a transaction block to execute the **update** command. The transaction descriptor associated with this block is acquired after the **go** command, and released after the insert has completed.

Because transaction descriptors consume memory that can be used by other Adaptive Server services, it is important that you use only enough descriptors to satisfy the maximum number of transactions that may be required at any given time.

Setting the Number of Transaction Descriptors

Once you have determined the number of transaction descriptors to use in your system, use **sp_configure** to set the value of **txn to pss ratio**. **txn to pss ratio** determines the total number of transaction descriptors available to the server. At boot time, this ratio is multiplied by the **number of user connections** parameter to create the transaction descriptor pool:

```
# of transaction descriptors = number of user
connections * txn to pss ratio
```

The default value, 16, ensures compatibility with earlier versions of Adaptive Server. Prior to version 12, Adaptive Server allocated 16 transaction descriptors for each user connection. In version 12, the number of simultaneous transactions is limited only by the number of transaction descriptors available in the server.

For example, to allocate 25 transaction descriptors for every user connection, use the command:

```
sp_configure 'txn to pss ratio', 25
```

You must reboot Adaptive Server for this change to take effect.

CHAPTER 3

Using Adaptive Server Transaction Coordination Services

This chapter describes how to configure and use Adaptive Server transaction coordination services. Topics include:

Overview of Transaction Coordination Services

The work of a local Adaptive Server transaction is sometimes distributed to remote servers that modify remote data. This can happen when a local transaction executes a remote procedure call (RPC) to update data in another Adaptive Server table, or when a local transaction modifies data in a remote table using Component Integration Services (CIS).

Prior to Adaptive Server version 12, local transactions that executed RPCs or updated data via CIS could not roll back the work done in remote Adaptive Servers. Moreover, the client executing the local transaction could not ensure that the remote work was actually committed if, for example, the remote server experienced a system failure.

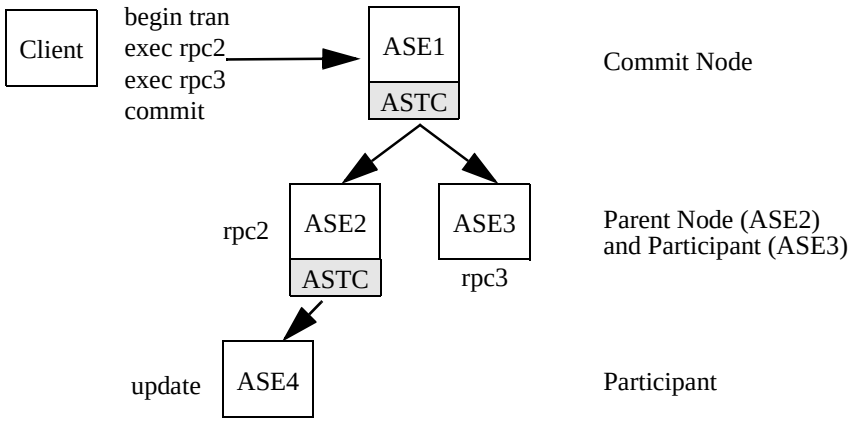
Adaptive Server version 12 provides services to propagate transactions to remote servers and coordinate the work of all servers, ensuring that all work is either committed or rolled back as a logical unit. With these transaction coordination services, Adaptive Server itself can act as a distributed transaction manager for transactions that update data in multiple Adaptive Servers.

Hierarchical Transaction Coordination

Because other Adaptive Servers involved in a distributed transaction may also coordinate remote participants, transactions can be further propagated to additional servers in a hierarchical manner. For example, in Figure 3-1, the client connected to ASE1 begins a transaction that executes an RPC on ASE2 and an RPC on ASE3. The coordination service for ASE1 propagates the transaction to ASE2 and ASE3.

Since ASE2 also has transaction coordination services enabled, it can propagate the transaction to additional remote participants. Here, ASE2 propagates the transaction to ASE4 where data is updated using CIS.

Figure 3-1: Hierarchical transaction coordination



In Figure 3-1, ASE1 is referred to as the **commit node** for the distributed transaction. When the transaction on ASE1 commits, the coordination service for ASE1 instructs ASE2 and ASE3 to prepare the transactions that it propagated to them. ASE3 indicates that its transaction is prepared when its local work is ready to be committed. ASE2 must complete its local work and instruct ASE4 to prepare its transaction. When the transactions are prepared in ASE2 and ASE4, the coordination service in ASE1 commits the original transaction. The instruction to commit subordinate transactions is then transmitted to ASE2, ASE3, and ultimately to ASE4, in the same manner as the instruction to prepare was transmitted.

X/Open XA-Compliant Behavior in DTP Environments

The X/Open XA protocol requires resource managers to provide coordination services for transactions that are propagated to remote resource managers. This requirement is made because the external transaction manager (and in some cases, the client originating the transaction) has no knowledge of when transactions are propagated to remote servers, and therefore cannot ensure that the remote transactions complete or abort as required.

The new transaction coordination service brings Adaptive Server, in its role as a resource manager, into full compliance with the X/Open XA protocol. Distributed transactions can be implicitly propagated to remote servers through RPCs and CIS, and Adaptive Server guarantees that the commit or rollback status of the global transaction is preserved in the remote servers it coordinates.

Requirements and Behavior

Adaptive Server transaction coordination services can ensure that the work of remote servers is logically committed or rolled back provided that each remote Adaptive Server is at version 12 or higher.

Transaction coordination services are transparent to the client executing the distributed transaction. When a local client transaction executes a RPC or updates data via CIS, the coordination service creates a new transaction name for the remote work and propagates that transaction to the subordinate, remote server. When the local client commits or rolls back the local transaction, Adaptive Server coordinates that request with each of the subordinate servers to ensure that the remote transactions are committed or rolled back as well.

The Adaptive Server transaction coordination service runs as one or more background tasks named "ASTC HANDLER," and can be viewed using **sp_who**. In systems using multiple Adaptive Server engines, the number of "ASTC HANDLER" processes (rounded down to the nearest whole number) is:

```
number of engines * 2/3
```

There can be a maximum of 4 "ASTC HANDLER" processes running on Adaptive Server.

The following output from **sp_who** shows a single "ASTC HANDLER":

```
                        sp_who
fid spid  status        loginame
   orignamе  hostname      blk_spid
   dbname cmd           block_xloid
----- ---------- ----------------- --------------
   ---------- ----------------- --------------
   ---------- ----------------- --------------
 0      1 running          sa
    sa    dtmsol1          0
    master SELECT          0
 0      2 sleeping         NULL
    NULL                   0
    master NETWORK HANDLER 0
 0      3 sleeping      NULL
    NULL                   0
    master DEADLOCK TUNE   0
 0      4 sleeping      NULL
    NULL                   0
    master MIRROR HANDLER0
 0      5 sleeping      NULL
```

```
    NULL                  0
    masterHOUSEKEEPER     0
0       6 sleeping      NULL
    NULL                  0
    master CHECKPOINT SLEEP   0
0       7 sleeping      NULL
    NULL  metin1_dtm    0
    sybsystemdb ASTC HANDLER  0
```

Configuring Participant Server Resources

By default, the transaction coordination service is always enabled. The System Administrator can enable or disable these services using the **enable xact coordination** configuration parameter. See the *System Administration Guide* for a complete description of this parameter.

The System Administrator must also ensure that Adaptive Server has the required resources to coordinate all of the RPCs and CIS updates that may be requested by transactions. Each time a transaction issues an RPC or CIS update, the transaction coordinator must obtain a free **DTX participant**. A DTX participant or "distributed transaction participant" is an internal memory structure that Adaptive Server uses to coordinate a transaction that has been propagated to a subordinate Adaptive Server. In Figure 3-1 ASE1 requires 3 free DTX participants, and ASE2 requires 2 free DTX participants. (In each case, a single DTX participant is used to coordinate the local work of the transaction that is propagated.

DTX participant resources remain in use by the coordinating Adaptive Server until the associated remote transaction has committed. This generally occurs some period of time *after* the initiating transaction has committed, since the initiating transaction commits as soon as all subordinate transactions have successfully prepared their work.

If no DTX participants are available, RPC requests and CIS update requests cannot proceed and the transaction is aborted.

number of dtx participants Parameter

The System Administrator can configure the total number of DTX participants available in Adaptive Server using the **number of dtx participants** configuration parameter. **number of dtx participants** sets the total number of remote transactions that the Adaptive Server transaction coordination service can propagate and coordinate at one time.

By default, Adaptive Server can coordinate 500 remote transactions. Setting **number of dtx participants** to a smaller number reduces the number of remote transactions that the server can manage. If no DTX participants are available, new distributed transactions will be unable to start. In-progress distributed transactions may abort if no DTX participants are available to propagate a new remote transaction.

Setting **number of dtx participants** to a larger number increases the number of remote transaction branches that Adaptive Server can handle, but also consumes more memory.

Optimizing *number of dtx participants* for Your System

During a peak period, use **sp_monitorconfig** to examine the use of DTX participants:

```
                    sp_monitorconfig "number of dtx participants"
Usage information at date and time: Jun 18 1999 9:00AM.
Name            # Free   # Active   % Active  # Max Ever Used   Re-used
 --------------  --------  ------   --------   ---------------   -------
 number of dtx 480  20  4.00     210          NA
 participants
```

If the *#Free* value is zero or very low, new distributed transactions may be unable to start due to a lack of DTX participants. Consider increasing the **number of dtx participants** value.

If the *#Max Ever Used* value is too low, unused DTX participants may be consuming memory that could be used by other server functions. Consider reducing the value of **number of dtx participants**.

Using Transaction Coordination Services in Heterogeneous Environments

When Adaptive Server propagates transactions to other version 12 or higher Adaptive Servers, it can ensure the integrity of the distributed transaction as a whole. However, the work of a local Adaptive Server transaction is sometimes distributed to remote servers that do not support version 12 transaction coordination services. This may occur when a transaction uses RPCs to update data in earlier Adaptive Server versions, or when CIS services are used to update data in non-Sybase databases. Under these circumstances the coordinating Adaptive Server cannot ensure that the work of remote servers is committed or rolled back with the original transaction.

strict dtm enforcement Parameter

In Adaptive Server version 12, the System Administrator can enforce or relax the requirement to have distributed transactions commit or roll back as a logical unit by setting the **strict dtm enforcement** configuration parameter.

Note You can also override the value of **strict dtm enforcement** using the session level **set** command with the **strict_dtm_enforcement** option.

---

strict dtm enforcement determines whether or not Adaptive Server transaction coordination services will strictly enforce the ACID properties of distributed transactions.

Setting **strict dtm enforcement** to 1 (on) ensures that transactions are propagated only to servers that can participate in Adaptive Server-coordinated transactions. If a transaction attempts to update data in a server that does not support transaction coordination services, Adaptive Server aborts the transaction.

In heterogeneous environments, you may want to make use of servers that do not support transaction coordination. This includes older versions of Adaptive Server and non-Sybase database stores configured using CIS. Under these circumstances, you can set **strict dtm enforcement** to 0 (off). This allows Adaptive Server to propagate transactions to legacy Adaptive Servers and other data stores, but does not ensure that the remote work of these servers is rolled back or committed with the original transaction.

Monitoring Coordinated Transactions and Participants

Adaptive Server tracks information about the status of work done in subordinate servers using data in the new system table, *sybsystemdb.dbo.syscoordinations*. See the Adaptive Server *Reference Manual* for a complete definition of this table.

The **sp_transactions** procedure also displays some data from the *syscoordinations* table for in-progress, remote transactions. See “Getting Information about Distributed Transactions” on page 26 for more information.

CHAPTER 4

DTM Administration and Troubleshooting

This chapter provides information about how to monitor, administer, and troubleshoot Adaptive Server DTM features. It includes the following sections:

Transactions and Threads of Control

Prior to Adaptive Server version 12, all of a transaction's resources were privately owned by a single server task. The server could not share a transaction with any task other than the one that initiated the transaction.

Adaptive Server version 12 provides native support for the "suspend" and "join" semantics used by X/Open XA-compliant transaction managers such as Encina and TUXEDO. Transactions may be shared among different threads of execution, or may have no associated thread at all.

When a transaction has no thread associated with it, it is said to be "detached." Detached transactions are assigned a *spid* value 0. You can see the transaction *spid* value in the new *master.dbo.systransactions* table, or in output from the new **sp_transactions** procedure. See "Getting Information about Distributed Transactions" on page 26 for more information.

Implications for System Administrators

Detached transactions are meant to persist in Adaptive Server, since the client application may want to reattach the original thread, or attach a new thread to the transaction. The System Administrator can no longer roll back a transaction by killing its associated *spid*, as a thread is not attached to the transaction.

Transactions in a detached state may also prevent the log from being truncated with the **dump transaction** command. In extreme circumstances, detached transactions can be rolled back by using the new **dbcc complete_xact** command to heuristically complete a transaction. See "Heuristically Completing Transactions" on page 34.

dtm detach timeout period Parameter

The system administrator can also specify a server-wide interval after which Adaptive Server automatically rolls back transactions that are in the detached state. **dtm detach timeout period** sets the amount of time, in minutes, that a distributed transaction branch can remain in the detached state. After this time has passed, Adaptive Server rolls back the detached transaction.

For example, to automatically rollback detached after 30 minutes, use the command:

```
sp_configure 'dtm detach timeout period', 30
```

Lock Manager Changes to Support Detached Transactions

Prior to Adaptive Server version 12, the lock manager could uniquely identify a transaction's locks by using the *spid* value of the transaction's thread. With the new transaction manager, transactions may be detached from their original threads, and have no associated *spid*. Moreover, multiple threads with different *spid* values must be able to share the same transaction locks to perform the work of a distributed transaction.

To facilitate these changes, the Adaptive Server version 12 lock manager uses a unique lock owner ID, rather than a *spid*, to identify transaction locks. The lock owner ID is independent from the *spid* that created the transaction, and it persists even when the transaction is detached from a thread. Lock owner IDs provide a way to support transactional locks when transactions have no associated threads, or when a new thread is attached to the transaction.

The lock owner ID is stored in the new *loid* column of *master.dbo.syslocks*. You can determine the *loid* value of a transaction by examining **sp_lock** or **sp_transactions** output.

Examining the *spid* and *loid* columns from **sp_transactions** output provides information about a transaction and its thread of control. A *spid* value of zero indicates that the transaction is detached from its thread of control. Non-zero *spid* values indicate that the thread of control is currently attached to the transaction.

If the *loid* value in **sp_transactions** output is even, then a local transaction owns the lock. Odd *loid* values indicate that an external transaction owns the lock.

See "Getting Information about Distributed Transactions" on page 26 for more information about **sp_transactions** output.

Getting Information about Distributed Transactions

Adaptive Server version 12 has a new system table, *master.dbo.systransactions*, which stores information about all server transactions. *systransactions* identifies each transaction and maintains information about the state of the transaction and its associated threads.

The new system procedure, **sp_transactions**, translates information from the *systransactions* and *syscoordinations* tables to display status conditions for active transactions.

Transaction Identification in *systransactions*

Adaptive Server stores transaction names in a column of varchar(255) (as compared to varchar(64) in previous server versions) to accommodate the length and format of transaction names supplied by different distributed transaction protocols. In the X/Open XA protocol, for instance, distributed transactions are assigned a transaction name consisting of both a global transaction ID (*gtrid*) and a branch qualifier. Within Adaptive Server, this information is combined in the *xactname* column of the *systransactions* table.

systransactions.xactname stores the names of both externally-created distributed transactions (defined by an X/Open XA transaction manager or MSDTC) and local server transactions. Clients defining local transactions can name those transactions anything they wish, within the confines of the varchar(255) column. Similarly, external transaction managers can use a variety of different formats to name a distributed transaction.

Transaction Keys

The transaction key, stored in the *xactkey* column of *systransactions*, acts as a unique internal handle to a server transaction. For local transactions, *xactkey* ensures that transactions can be distinguished from one another, even if the transaction name is not unique to the server.

Beginning with Adaptive Server version 12, all system tables refer to *systransactions.xactkey* to uniquely identify a transaction. The *sysprocesses* and *syslogshold* tables are the only exceptions to this rule—they reference *systransactions.xactname* and truncate the value to a length of varchar(64) (for *sysprocesses*) and varchar(67) (for *syslogshold*), to maintain backward compatibility with earlier Adaptive Server versions.

Viewing Active Transactions with *sp_transactions*

The **sp_transactions** procedure translates information from *systransactions* and *syscoordinations* to provide information about active transactions. When used without keywords, **sp_transactions** displays information about all active transactions:

```
                sp_transactions
xactkey                    type        coordinator starttime
 state            connection dbid   spid   loid
 failover                 srvname                        namelen
 xactname
 ---------------------------- ----------- -----------
 -------------------
 ---------------- ---------- ------ ------ -----------
 ------------------------- ------------------------------ -------
 ------------------------------------------------------------------
 0x00000b1700040000dd6821390001 Local       None        Jun 1 1999 3:47PM
 Begun            Attached        1   1      2
 Resident Tx              NULL                                17
 $user_transaction
 0x00000b1700040000dd6821390001 Remote      ASTC        Jun 1 1999 3:47PM
 Begun            NA              0   8      0
 Resident Tx              caserv2                         108

 00000b1700040000dd6821390001-aa01f04ebb9a-00000b1700040000dd6821390001-
aa01f04ebb9a-caserv1-caserv1-0002
```

Identifying Local, Remote, and External Transactions

The “type” column indicates whether the transaction is local, remote, or external. Local transactions execute on the local server (the server on which you ran **sp_transactions**). Local transactions have a null value in the “srvname” column, since the transaction takes place on the current server.

For remote transactions, **sp_transactions** lists the name of the server executing the transaction under the “srvname” column. The **sp_transactions** output above shows a remote transaction executing on the server named “caserv2.”

External transactions indicate that the transaction is coordinated by an external transaction coordinator, such as CICS, Encina, or the “ASTC HANDLER” process of another Adaptive Server. External transactions also have a null value in the “srvname” column.

Identifying the Transaction Coordinator

The “coordinator” column indicates the method or protocol used to manage a transaction. In the output above, the local transaction “$user_transaction” does not have an external coordinator. The remote transaction taking place on “caserv2” has the coordinator value “ASTC.” This indicates that the transaction is coordinated using native Adaptive Server coordination services, as described under Chapter 3, “Using Adaptive Server Transaction Coordination Services”.

See **sp_transactions** in the Adaptive Server *Reference Manual* for a complete list and description of possible coordinator values.

Viewing the Transaction Thread of Control

The “spid” column displays the process ID of the process attached to the transaction (or 0 if the transaction is detached from its thread of control). For local transactions, the *spid* value indicates a process ID running on the local server. For remote transactions, the *spid* indicates the process ID of a task running on the indicated remote server. The output above shows a *spid* value of 8 running on the remote server, “caserv2.”

Understanding Transaction State Information

The “state” column displays information about the current state of each transaction. At any given time, a local or external transaction may be executing a command, aborted, committed, and so forth. Additionally, distributed transactions can be in a prepared state, or can be heuristically completed or rolled back.

The “connection” column displays information about the state of the transaction’s connection. You can use this information to determine whether a transaction is currently attached to or detached from a process. Transactions in X/Open XA environments may become detached from their initiating process, in response to requests from the transaction manager.

See **sp_transactions** in the Adaptive Server *Reference Manual* for a complete list and description of possible coordinator values.

Limiting sp_transactions Output to Specific States

You can use **sp_transactions** with the **state** keyword to limit output to the specified transaction state. For example:

```
sp_transactions "state", "Prepared"
```

displays information only for distributed transactions that have been prepared.

Transaction Failover Information

The “failover” column displays special information for servers operating in high availability environments. In high availability environments, prepared transactions may be transferred to a secondary companion server if the original server experiences a critical failure. The “failover” column can display three possible failover states that indicate how and where the transaction is executing:

- “Resident Tx” is displayed under normal operating conditions, and on systems that do not utilize Adaptive Server high availability features. “Resident Tx” means that the transaction was started and is executing on a primary Adaptive Server.
- “Failed-over Tx” is displayed after there has been a failover to a secondary companion server. “Failed-over Tx” means that a transaction originally started on a primary server and reached the prepared state, but was automatically migrated to the secondary companion server (for example, as a result of a system failure on the primary server). The migration of a prepared transaction occurs transparently to an external coordinating service.
- “Tx by Failover-Conn” is also displayed after there has been a failover to a secondary companion server. “Tx by Failover-Conn” indicates that the application or client attempted to start the transaction on a primary server, but the primary server was not available due to a connection failover. When this occurs, the transaction is automatically started on the secondary companion server, and the transaction is marked “Tx by Failover-Conn.”

See “Using Sybase Failover in a High Availability System” in this document for more information about Adaptive Server failover features.

Determining the Commit Node and gtrid with *sp_transactions*

Using **sp_transactions** with the **xid** keyword displays the commit node, parent node, and *gtrid* of a particular transaction, in addition to the output described under “Viewing Active Transactions with sp_transactions” on page 27. This form of **sp_transactions** requires that you specify a particular transaction name. For example:

```
sp_transactions "xid", "00000b1700040000dd6821390001-aa01f04ebb9a-
00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-caserv1-0002"
```

```
xactkey                         type        coordinator starttime
 state             connection dbid   spid   loid
 failover                   srvname                         namelen
 xactname
 commit_nodeparent_node
 gtrid
 ------------------------------ ----------- -----------
 --------------------
 ----------------- ---------- ------ ------ -----------
 -------------------------- ------------------------------ -------
 ------------------------------------------------------------------
 -------------
 -------------
 -------------
 0x00000b2500080000dd6821960001 External    ASTC        Jun 1 1999 3:47PM
 Begun             Attached        1      8         139
 Resident Tx               NULL                                108

 00000b1700040000dd6821390001-aa01f04ebb9a-00000b1700040000dd6821390001-
aa01f04ebb9a-caserv1-caserv1-0002

 caserv1  caserv1
 00000b1700040000dd6821390001-aa01f04ebb9a
```

Commit and Parent Nodes

For distributed transactions coordinated by Adaptive Server, the “commit node” column lists the name of the server that executes the topmost branch of the distributed transaction. This transaction determines the commit or rollback status for all branches of the transaction. See “Hierarchical Transaction Coordination” on page 14 for more information.

The “parent node” column lists the name of the server that initiated the transaction. In the **sp_transactions** output above, the “commit node” and “parent node” columns list the same server, “caserv1.” This indicates that the distributed transaction originated on “caserv1”, and “caserv1” propagated a branch of the transaction to the current server.

Global Transaction ID

The “gtrid” column displays the global transaction ID for distributed transactions coordinated by Adaptive Server. Transaction branches that are part of the same distributed transaction share the same *gtrid*. You can use a specific *gtrid* with the **sp_transactions gtrid** keyword to determine the state of other transaction branches running on the current server. This is useful for System Administrators who must determine whether a particular branch of a distributed transaction should be heuristically committed or rolled back. See “Determining the Commit Status for Adaptive Server Transactions” on page 36 for an example that uses **sp_transactions** with the **gtrid** keyword.

Note For transactions coordinated by an X/Open XA-compliant transaction manager, MSDTC, or SYB2PC, the *gtrid* column shows the full transaction name supplied by the external coordinator.

Crash Recovery Procedures for Distributed Transactions

During crash recovery, Adaptive Server must resolve distributed transactions that it discovers in the prepared state. The method used to resolve prepared transactions depends on the coordination method or coordination protocol used to manage the distributed transaction.

Note The following crash recovery procedures are not performed during normal database recovery for **load database** or **load transaction** commands. If **load database** or **load transaction** applies any transactions that prepared or in-doubt, Adaptive Server aborts those transactions before bringing the associated database online.

Transactions Coordinated with MSDTC

Prepared transactions that were coordinating using MSDTC are rolled forward or backward depending on the commit status of the master transaction. During recovery, Adaptive Server initiates contact with MSDTC to determine the commit status of the master transaction, and commits or rolls back the prepared transaction accordingly. If it cannot contact MSDTC, the recovery procedure waits until contact is established. Further recovery does not take place until Adaptive Server has established contact with MSDTC.

Transactions Coordinated by Adaptive Server or X/Open XA

During crash recovery, Adaptive Server may also encounter prepared transactions that were coordinated using Adaptive Server transaction coordination services or the X/Open XA protocol. Upon encountering these transactions, the local server must wait for the coordinating Adaptive Server or the external transaction coordinator to initiate contact and indicate whether the prepared transaction should commit or roll back.

To speed the recovery process, Adaptive Server restores each of these transactions to their condition prior to the failure. The transaction manager creates a new transaction with the original transaction ID, and the lock manager applies locks to protect data that the original transaction was modifying. The restored transaction remains in a prepared state but is detached, having no thread associated with it.

Once the transaction's coordinator contacts Adaptive Server, the transaction manager can commit or roll back the transaction.

Using this recovery mechanism, the server can bring a database online even when the coordinating Adaptive Server or external transaction manager has not yet attempted to resolve the prepared transaction. Other clients and transactions can resume work on the local data, since the prepared transaction holds the locks it did prior to recovery. The prepared transaction itself is ready to commit or roll back once contacted by its coordinator.

When the controlling Adaptive Server or external transaction manager cannot complete the transaction, the System Administrator can heuristically complete the transaction to free its locks and transaction resources. See "Heuristically Completing Transactions" on page 34 for more information.

Transactions Coordinated with SYB2PC

Prepared transactions that were coordinated using the SYB2PC protocol are rolled forward or backward depending on the commit status of the master transaction. During recovery, Adaptive Server initiates contact with the commit service to determine the commit status of the master transaction, and commits or rolls back the prepared transaction accordingly. If it cannot contact the commit service, Adaptive Server does not bring the database online. However, Adaptive Server does proceed to recover other databases in the system.

This recovery method was used for SYB2PC transactions in earlier Adaptive Server versions and is unchanged with Adaptive Server version 12.

Heuristically Completing Transactions

Adaptive Server version 12 includes the new **dbcc complete_xact** command to facilitate heuristic completion of transactions. **dbcc complete_xact** resolves a transaction by either committing or rolling back its work, freeing whatever resources the transaction was using.

dbcc complete_xact is provided for those cases where only the System Administrator can properly resolve a prepared transaction, or for when the System Administrator must resolve a transaction without waiting for the transaction's coordinator.

For example, in Figure 3-1 on page 15, heuristic completion may be considered if all remote Adaptive Servers have prepared their transactions, but the network connection to ASE1 was *permanently lost*. The remote Adaptive Servers will maintain their transactions in a prepared state until contacted by the coordination service from ASE1. In this case, only the System Administrator for ASE2, ASE3, and ASE4 can properly resolve the prepared transactions. Heuristically completing the prepared transaction in ASE3 frees up transaction and lock resources, and records the commit status in *systransactions* for later use by the transaction coordinator. Heuristically completing the transaction in ASE2 also completes the transaction propagated to ASE4.

Completing Prepared Transactions

Warning! Heuristically completing a prepared transaction can cause inconsistent results for an entire distributed transaction. The System Administrator's decision to heuristically commit or roll back a transaction may contradict the decision made by the coordinating Adaptive Server or transaction protocol.

Before heuristically completing a transaction, the System Administrator should make every effort to determine whether the coordinating Adaptive Server or transaction protocol decided to commit or roll back the distributed transaction (see "Determining the Commit Status for Adaptive Server Transactions" on page 36).

By using **dbcc complete_xact**, the System Administrator forces Adaptive Server to commit or roll back a branch of a distributed transaction. After heuristically completing a prepared transaction, Adaptive Server records the transaction's commit status in *master.dbo.systransactions* so that the transaction's coordinator—Adaptive Server, MSDTC, or an X/Open XA transaction manager—can know whether the transaction was committed or rolled back.

Adaptive Server propagates the command to heuristically commit or abort a transaction to any participant servers that it coordinated for the transaction branch. For example, if in Figure 3-1 on page 15 you heuristically commit the transaction on ASE2, ASE2 propagates the command to ASE4 so that the transaction on ASE4 also commits.

dbcc complete_xact requires that you supply an active transaction name and desired outcome for the transaction. For example, the following command heuristically commits a transaction:

```
dbcc complete_xact "00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-
aa01f04ebb9a-caserv1-caserv1-0002", "commit"
```

Forgetting Heuristically Completed Transactions

When the System Administrator heuristically completes a prepared transaction, Adaptive Server maintains information about the transaction's commit status in *master.dbo.systransactions*. This information is maintained so external transaction coordinators can detect the presence of heuristically completed transactions.

If the external coordinator is another Adaptive Server, the server examines the commit status and logs a warning message if the heuristic completion conflicts with the commit status of the distributed transaction. After examining the commit status, the coordinating Adaptive Server clears the commit status information from *systransactions*.

If the external coordinator is an X/Open XA-compliant transaction manager, the transaction manager does not log warning message when the heuristic completion conflicts with the distributed transaction. However, X/Open XA-compliant transaction managers clear the commit status information from *systransactions*.

Manually Clearing the Commit Status

dbcc forget_xact purges the commit status of a heuristically completed transaction from *systransactions*. It can be used when the System Administrator does not want the coordinating service to have knowledge that a transaction was heuristically completed, or when an external coordinator will not be available to clear information from *systransactions*.

See **dbcc** in the Adaptive Server *Reference Manual* for more information about using **dbcc forget_xact**.

Completing Transactions That Are Not Prepared

dbcc complete_xact can also be used to roll back Adaptive Server-coordinated transactions that have not yet reached the prepared state. Heuristically rolling back a transaction that has not yet been prepared does not pose a risk to the distributed transaction, since the coordinating server can recognize that the transaction failed to prepare its work. Under these circumstances, the coordinating Adaptive Server can roll back the entire distributed transaction to preserve consistency.

When you heuristically roll back an Adaptive Server transaction that has not yet been prepared, Adaptive Server *does not* record the heuristic roll back in *systransactions*. Instead, an informational message is printed to the screen and recorded in the server's error log.

Determining the Commit Status for Adaptive Server Transactions

If the distributed transaction branch you want to commit or roll back is coordinated by Adaptive Server, you can use **sp_transactions** to determine the commit status of the distributed transaction. To do so, complete the following steps.

Note These steps cannot be used with distributed transactions that are coordinated by the X/Open XA protocol, MSDTC, or SYB2PC.

---

1 In the server that is executing the transaction branch you want to complete, use **sp_transactions** with the **xid** keyword to display information about the transaction. Record the commit node and *gtrid* of the transaction. For example:

```
                    sp_transactions "xid",
                    "00000b1700040000dd6821390001-aa01f04ebb9a-
                    00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-
                    caserv1-0002"
xactkey                         type        coordinator starttime
 state              connection dbid   spid   loid
 failover                   srvname                        namelen
 xactname
 commit_nodeparent_node
 gtrid
 ------------------------------ ----------- -----------
 --------------------
 ----------------- ---------- ------ ------ -----------
 ------------------------- ------------------------------ -------
 ------------------------------------------------------------------
 -------------
 -------------
 -------------
 0x00000b2500080000dd6821960001 External    ASTC       Jun 1 1999 3:47PM
 Begun             Attached        1      8         139
 Resident Tx                NULL                                108

 00000b1700040000dd6821390001-aa01f04ebb9a-
00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-caserv1-0002

 caserv1sfserv
 00000b1700040000dd6821390001-aa01f04ebb9a
```

In this example, the commit node for the distributed transaction is “caserv1” and the *gtrid* is “00000b1700040000dd6821390001-aa01f04ebb9a”.

2 Log on to the server indicated by the commit node. For example:

```
isql -Usa -Psa_password -Scaserv1
```

3 Use **sp_transactions** with the **gtrid** keyword to determine the commit status of the distributed transaction having the *gtrid* obtained in step 1:

```
                    sp_transactions "gtrid",
                    "00000b1700040000dd6821390001-aa01f04ebb9a"
 xactkey                         type        coordinator starttime
 state              connection dbid   spid   loid
 failover                   srvname                        namelen
 xactname
 commit_node
 parent_node
 ------------------------------ ----------- -----------
```

```
--------------------
---------------- ---------- ------ ------ -----------
------------------------- ------------------------------ -------
-----------------------------------------------------------------
-------------
-------------
0x00000b1700040000dd6821390001 Local      None       Jun 1 1999 3:47PM
Committed Attached        1      1           2
Resident Tx                NULL                                17
$user_transaction

caserv1
caserv1
```

In this example, the local transaction with the specified *gtrid* has committed, as indicated by the "state" column. The System Administrator should heuristically *commit* the prepared transaction examined in step 1.

4 Using an account with System Administrator privileges, log on to the server that is executing the transaction branch you want to complete:

```
isql -Usa -Psa_password -Ssfserv
```

5 Use **dbcc complete_xact** to commit the transaction. In this example, the System Administrator should use the **commit** keyword to maintain consistency with the distributed transaction:

```
dbcc complete_xact "00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-
aa01f04ebb9a-caserv1-caserv1-0002", "commit"
```

Index

I

K

L

M

N

O

P

R

S

T

U

X