Oracle® Services for Microsoft Transaction Server

Developer's Guide

19c for Microsoft Windows

E96617-01
May 2019

Oracle Services for Microsoft Transaction Server Developer's Guide, 19c for Microsoft Windows

E96617-01

Primary Author: Maitreyee Chaliha

Contributing Authors: Janis Greenberg, Patricia Huey, Mark Kennedy, Roza Leyderman, Janelle Simmons, Alex Keh, Valarie Moore, Vivek Raja, Eric Wang, Yong Hu

Contents

3 Managing Recovery Scenarios

4 Programming with Microsoft Transaction Server and an Oracle Database

List of Figures

List of Tables

Preface

This manual explains how to install, configure, use, and administer Oracle Services for Microsoft Transaction Server that apply to operating systems. It covers the features of Oracle Database software that apply to the Windows 2000, Windows XP, and Windows Server 2003 operating systems.

This preface contains these topics:

Audience

This guide is intended for anyone who performs the following tasks:

- Uses component object model (COM) components with Microsoft Transaction Server
- Registers COM components as transactional and has Microsoft Transaction Server control the transaction
- Uses client-side connection pooling in Microsoft Transaction Server
- Uses .NET applications with Oracle Services for Microsoft Transaction Server to access Oracle Database instances.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Reference*
- *Oracle Provider for OLE DB Developer's Guide for Microsoft Windows*
- *Oracle Data Provider for .NET Developer's Guide for Microsoft Windows*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database Platform Guide for Microsoft Windows*

For information about Oracle error messages, see *Oracle Database Error Messages*. Once you find the specific range, you can search for the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

Many of the examples in this book use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN)

`http://www.oracle.com/technetwork/index.html`

For the latest version of the Oracle documentation, including this guide, visit

`http://www.oracle.com/technetwork/indexes/documentation/index.html`

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Services for Microsoft Transaction Server Developer's Guide

This preface contains:

Changes in Oracle Services for Microsoft Transaction Server in ODAC 12*c* Release 3

The following are changes in *Oracle Services for Microsoft Transaction Server Developer's Guide* for ODAC 12*c* Release 3.

New Features

The following features are new in this release:

- Distributed Transactions without `Oracle.ManagedDataAccessDTC.dll`

  The `Oracle.ManagedDataAccessDTC.dll` assembly is no longer required for distributed transaction applications running in .NET Framework 4.5.2 or higher and ODP.NET, Managed Driver. Upon ODP.NET installation, `Oracle.ManagedDataAccessDTC.dll` is no longer placed into the Global Assembly Cache (GAC). For applications that use .NET Framework 4.5.1 or earlier, `Oracle.ManagedDataAccessDTC.dll` needs to be placed either in the application directory or in the GAC.

Changes in Oracle Database 12*c* Release 1 (12.1)

The following are changes in *Oracle Services for Microsoft Transaction Server Developer's Guide* for Oracle Database 12*c* Release 1 (12.1).

New Features

The following features are new in this release:

- OraMTS Services Run as Oracle Home User

  OraMTS services now operate as a specified, lower privileged user (the Oracle Home User). A utility is provided for cases where the service must be created

manually rather than by the installer. Each Oracle Home now has its own OraMTS service.

> **See Also:**
>
> "Manually Creating an Oracle MTS Recovery Service."

1

Using Microsoft Transaction Server with Oracle Database

These topics describe Microsoft Transaction Server and Oracle Database integration.

- Microsoft Transaction Server Overview
- Microsoft Transaction Server and Oracle Integration Overview
- Getting Started with Microsoft Transaction Server and Oracle

Microsoft Transaction Server Overview

Microsoft Transaction Server is a proprietary component object model (COM) transaction processing system that runs on an Internet or network server. Microsoft Transaction Server deploys and manages application and database transaction requests on behalf of a client computer. Microsoft Transaction Server provides:

- ActiveX/distributed component object model (DCOM) programming model to develop distributed applications and a runtime environment in which to deploy these applications.
- Atomicity_ Consistency_ Isolation_ and Durability (ACID) properties for components in transactions.
- Access to performance-enhancing features such as component caching and database connection pooling.

Microsoft Transaction Server is a component of the three-tiered, server-centric architecture model. This model lets you separate the presentation, business logic, and data elements of applications onto different computers connected in a network. Microsoft Transaction Server functionality is also implemented in COM+ and Enterprise Services. Oracle Services for Microsoft Transaction Server, or OraMTS, support Microsoft Transaction Server, COM+, and Enterprise Services.

> **See Also:**
>
> Microsoft documentation for additional information about Microsoft Transaction Server

Microsoft Transaction Server and Oracle Integration Overview

Without any special integration, you can deploy applications that were created using Win32, Win64, COM, or Microsoft .NET with a Microsoft Transaction Server that

connects to an Oracle Database. To use either of the following features, however, you must install Oracle Services for Microsoft Transaction Server (OraMTS):

- Register the Win32, Win64, COM or .NET application as transactional and have Microsoft Transaction Server control the transaction. You can do this by using the Properties dialog box of the component in the Microsoft Management Console Explorer.
- Use client-side connection pooling in Microsoft Transaction Server.

After you have installed Oracle Services for Microsoft Transaction Server, an Oracle MTS Recovery Service is also automatically installed on the same computer. The Oracle MTS Recovery Service helps in the recovery of in-doubt transactions left in Oracle Database instances that originated from this computer. On each connected database:

- Create the Microsoft Transaction Server administrator user account.
- Schedule a database-level transaction recovery job.

This enables the database to participate in Microsoft Transaction Server-started transactions.

Create the COM component with any of the following Oracle products:

- Oracle Data Provider for .NET (ODP.NET).
- Oracle Open Database Connectivity (ODBC) Driver
- Oracle Provider for OLE DB
- Oracle Call Interface (OCI)

Oracle Services for Microsoft Transaction Server Support for DTC

Oracle Services for Microsoft Transaction Server works with Microsoft Distributed Transaction Coordinator (DTC), which is part of the Windows operating system. DTC implements a two-phase commit protocol that makes sure that the transaction outcome is consistent across all data resources involved in a transaction.

Distributed Transactions on Real Application Clusters (Oracle RAC)

With Oracle Database Release 11.1, the database now redirect all the branches of a distributed transaction to a single Oracle RAC instance automatically. Previously developers needed to manually manage this process, individually redirecting all the branches to a single Oracle RAC instance.

> **See Also:**
>
> *Oracle Real Application Clusters Administration and Deployment Guide* to learn more about distributed transactions in Real Application Clusters.

Promotable Local Transactions

Promotable local transactions allow all transactions to remain local until more than one database is brought into the transaction, at which point, they are promoted to distributed transactions.

The flexibility of the promotable transaction feature ensures more efficient resource usage for transactional applications. Distributed transactions require significant overhead versus local transactions. Therefore, local transactions are preferred if only one database is used. At design-time, it may not be known when transactions are local or distributed. Prior to this feature, developers always had to use distributed transactions, even if local ones occurred most of the time, leading to unnecessary resource usage.

This feature is supported with Oracle Database 11*g* Release 1 and higher. Earlier database versions and other resource managers can participate in a promotable transaction as long as the first connection is to an Oracle Database 11*g* Release 1 data source or higher.

> **See Also:**
>
> *Oracle Data Provider for .NET Developer's Guide for Microsoft Windows* for more information on `System.Transactions` support

Read-Committed and Serializable Transactions

Oracle Services for Microsoft Transaction Server supports distributed transactions set to a serializable or read-committed isolation level.

Getting Started with Microsoft Transaction Server and Oracle

You are now ready to use Microsoft Transaction Server with a database. To get started quickly, follow these steps:

1. Install the Oracle and Microsoft products required for Microsoft Transaction Server and database integration.

   See Installing Oracle Services for Microsoft Transaction Server .

2. Create the Microsoft Transaction Server administrator user account.

   See Managing Recovery Scenarios.

3. Schedule a Microsoft Transaction Server transaction recovery job.

   See Managing Recovery Scenarios.

4. Create Microsoft Transaction Server-hosted applications.

   See Programming with Microsoft Transaction Server and an Oracle Database for instructions on using OCI, Oracle ODBC Driver, or Oracle Provider for OLE DB with COM-based applications.

5. Learn about using Microsoft Transaction Server on the different Windows operating systems.

2

Installing Oracle Services for Microsoft Transaction Server

These topics describe installation requirements for the Microsoft Transaction Server and Oracle Database environment.

- Installation Introduction
- Installation Requirements for Microsoft Transaction Server
- Using Oracle Services for MTS with Oracle Data Provider for .NET_ Managed Driver
- Manually Creating an Oracle MTS Recovery Service

Installation Introduction

You can install OraMTS as part of the Oracle Database Client. OraMTS service will be created for the Oracle Home you installed. Beginning in Oracle 12*c*, each Oracle Home has its own OraMTS recovery Windows service. Also, beginning in Oracle 12*c*, the OraMTS service runs as the Oracle Home user.

Oracle Home User is the owner of Oracle Services that run from Oracle Home and cannot be changed after installation. It can be a Windows built-in account or a Windows User Account. For enhanced security, Oracle recommends choosing the standard Windows User Account as Oracle Home User for Oracle Database installations rather than a Windows built-in account. The primary purpose of Oracle Home User is to run Windows services with Windows User Account. However, this user account (Oracle Home User) must be a low-privileged user account that should not be used for database administration. This ensures that Oracle Database services running under Oracle Home User have only the minimal privileges required to run Oracle products. The Windows User Account can be a Local User, a Domain User, or a Managed Services Account.

In some cases, such as a software-only install, clone cycles, or an `AddNode` operation on the database, you may need to manually create the OraMTS service.

This topic describes the Oracle and non-Oracle products you must install for OraMTS. Additional installation requirement is as follows. For Oracle Data Provider for .NET (ODP.NET) cluster configurations (or any failover configuration), install Microsoft Transaction Server on the node running the Microsoft Distributed Transaction Coordinator (MS DTC) component. This ensures that the Oracle MTS Recovery Service migrates with the client application during failover. You can configure this when scheduling recovery transactions.

> **See Also:**
>
> - Manually Creating an Oracle MTS Recovery Service
> - *Oracle Database Platform Guide for Microsoft Windows*

Installation Requirements for Microsoft Transaction Server

The Windows computer where Microsoft Transaction Server is installed has the following product requirements:

Oracle Products

- OraMTS
- Oracle Database Client, included automatically with the OraMTS installation.
- A data access driver that uses Oracle Services for MTS, such as:
  - Oracle Data Provider for .NET (ODP.NET)
  - Oracle Open Database Connectivity (ODBC) Driver
  - Oracle Provider for OLE DB
  - Oracle Call Interface (OCI)
- Access to an Oracle Database 11*g* Release 2 or higher version

> **Note:**
>
> - Oracle ODBC Driver, ODP.NET, Oracle Provider for OLE DB, and OCI are only required if you are building or using components with which they are required.
> - Depending on the installation, you are prompted to enter the port number on which the Oracle MTS Recovery Service will listen for requests to resolve in-doubt transactions

Non-Oracle Products

- Windows operating system

  > **See Also:**
  >
  > *Oracle Database Installation Guide for Microsoft Windows*

- Microsoft Distributed Transaction Coordinator, which is part of the Windows operating system

Distributed Transactions

A distributed transaction includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.

ODP.NET, Managed and Unmanaged Drivers integrate with Microsoft Distributed Transaction Coordinator (MSDTC) and Oracle databases. MSDTC coordinates with all the resource managers that are enlisted to the same `System.Transactions` object, to perform a 2-phrase commit or rollback atomically. With that, Oracle distributed transactions can then be committed or rolled back across networks properly.

ODP.NET only supports the read committed isolation level for distributed transactions.

> **Note:**
>
> ODP.NET Core does not support Distributed Transactions.

Microsoft Distributed Transaction Coordinator Integration

Managed ODP.NET includes a native fully managed implementation that supports integration with MSDTC for distributed transactions enlistments, commits, and rollbacks.

For unmanaged ODP.NET, Oracle Services for Microsoft Transaction (OraMTS) provides MSDTC integration that allows client components to participate in Oracle database distributed transactions. OraMTS act as a proxy for the Oracle database to MSDTC to ensure that Oracle distributed database transactions commit or rollback together with the rest of the distributed transaction.

Unmanaged ODP.NET can use either a managed or unmanaged OraMTS implementation. Oracle recommends using the managed OraMTS for applications requiring high availability from Oracle RAC or Data Guard.

If a failure occurs in a distributed transaction, such as a network failure or server hardware failure, then it can leave an in-process transaction in-doubt. Both managed and unmanaged ODP.NET use the OraMTS recovery service to resolve these in-doubt transactions. This recovery service runs as a Windows service. Administrators must install and configure the OraMTS Windows recovery service to manage recovery scenarios whether they use managed or unmanaged ODP.NET. Only one recovery service is needed per machine.

It is required to install the OraMTS Recovery Service on all the client machines where ODP.NET is running and participating in MSDTC. As a machine may have multiple IP addresses, administrators for managed ODP.NET applications can specify the host machine name or IP address that has the running recovery service in the application's .NET configuration file. ODP.NET, Unmanaged Driver resolves the IP/ machine name for the recovery service automatically.

Table 2-1 Supported ODP.NET Type and .NET Framework Version for Distributed Transaction

ODP.NET Type	.NET Framework Version	Distributed Transaction Support
Managed	4.5.2 and higher	.NET Framework's native managed implementation for distributed transactions.
Unmanaged	4.5.2 and higher	OraMTS (default) or managed OraMTS implementation. Oracle recommends using managed OraMTS for unmanaged ODP.NET applications requiring high availability from Oracle RAC or Data Guard.
Unmanaged	3.5	OraMTS

Note:

.NET Framework 4.5.1 and lower within the .NET Framework 4 family are no longer supported by Microsoft and Oracle.

See Also:

Manually Creating an Oracle MTS Recovery Service in *Oracle Services for Microsoft Transaction Server Developer's Guide for Microsoft Windows*.

ODP.NET, Managed Driver Setup

This section explains the setup and configuration steps required for using distributed transactions with ODP.NET, Managed Driver. Follow these steps to configure distributed transactions in these .NET Framework versions:

1. Create and setup the OraMTS recovery service or make sure an existing recovery service is running.
2. Set the value of `OMTSRECO_PORT` in the .NET configuration to specify the port number that the OraMTS recovery service is running.

ODP.NET, Unmanaged Driver Setup

This section explains the setup and configuration steps required for using distributed transactions with ODP.NET, Unmanaged Driver.

For .NET Framework 4.5.2 and higher, ODP.NET, Unmanaged Driver includes managed OraMTS in its assembly. OraMTS is the default option for ODP.NET, Unmanaged Driver, to ensure implementation continuity during upgrades. However, Oracle recommends the managed OraMTS option when using any high availability FAN operations (HA Events = true) with Oracle Real Application Clusters or Oracle Data Guard. The managed OraMTS option support high availability functionality, while the traditional OraMTS does not.

Applications can set OraMTS (default) or managed OraMTS usage through the `UseOraMTSManaged` parameter in the .NET configuration file.

Install and configure OraMTS, including its recovery service to use traditional OraMTS implementation for ODP.NET, Unmanaged Driver.

To set managed OraMTS up, perform the following steps:

1. Set `UseOraMTSManaged` to `true` in the .NET configuration file.
2. Create and setup the OraMTS recovery service or make sure an existing recovery service is running.

> **See Also:**
>
> distributedTransaction section for more information about .NET configuration setup

Manually Creating an Oracle MTS Recovery Service

Starting with Oracle Database 12*c*, Oracle Database on Windows software includes the `oramtsctl.exe` utility to enable manual creation of the OraMTS recovery service. ODP.NET, Managed and Unmanaged Drivers can both opt to use fully managed MS DTC solutions, which eliminates the need for OraMTS. However, in all cases, the OraMTS recovery service is still required to be setup. Every Oracle client machine that uses MS DTC must have one OraMTS recovery service configured. Multiple Oracle applications on the same machine can share using that one service. When run, the service will be created for the current Oracle home and runs as the Oracle Home User.

To manually create an Oracle MTS Recovery Service:

1. Install an Oracle Database.
2. Open the command prompt.
3. Run this command:

```
oramtsctl -new
```

To delete an Oracle MTS Recovery Service run:

```
oramtsctl -new

oramtsctl -delete
```

Usage

```
oramtsctl <-new|-delete|-start|-stop|-status|-config|-passwd|-trace|-trcdir>
          [-host    <hostname|IP_address> ...]
          [-port    <number>]
          [-ip      <IP_address>]
          [-dtchost <DTC_hostname>]
          [-cluster <on|off>]
          [-default]

 -new
```

Create and start a new `Oracler1MTSRecoveryService` service.

```
-delete [Oracle_home_directory]
```

Delete `Oracler1MTSRecoveryService` of current home or specified home.

```
-start
```

Start `Oracler1MTSRecoveryService` service.

```
-stop
```

Stop `Oracler1MTSRecoveryService` service.

```
-status Query Oracler1MTSRecoveryService
```

service status.

```
-config [-port <number>] [-ip <IP_address>] [-dtchost <DTC_hostname>]
        [-cluster <on|off>] [-default]
```

Show or set `Oracler1MTSRecoveryService` service configuration.

Use -default to reset `Oracler1MTSRecoveryService` configuration.

```
-passwd
```

Update `Oracler1MTSRecoveryService` service user password.

```
-trace  [0-5]
```

Show traces or set trace level. Set level 0 to turn off tracing.

```
-trcdir [trace_directory]
```

Show or set trace directory.

```
-host   <hostname|IP_address> ...
```

Execute operation on host(s) identified by name(s) or IP address(es).

Local host is used if option not specified.

```
-port   <number>
```

Execute `-new` or `-config` with this option to specify a service port.

Service port is configured automatically if option not specified.

```
-ip     <IP_address>
```

Execute `-new` or `-config` with this option to specify an IP address.

```
-default
```

Execute `-new` or `-config` with this option to set default configuration.

```
oramtsctl -new -host host1 host2.domain.com host3 -port 2033
```

Examples:

Install the service with automatic configuration:

```
oramtsctl -new
```

Install the service on port 2032:

```
oramtsctl -new -port 2032
```

Configure the service to use port 2033:

```
oramtsctl -config -port 2033
```

Update the service user password:

```
oramtsctl -passwd
```

Install the service on several hosts and use uniform port 2033:

```
oramtsctl -new -host host1 host2.domain.com host3 -port 2033
```

3
Managing Recovery Scenarios

These topics describe how to create and schedule Microsoft Transaction Server-related Oracle transaction recovery.

- Microsoft Transaction Server Configuration Requirements
- Microsoft Transaction Server Transaction Recovery Overview
- Scheduling Automatic Microsoft Transaction Server Transaction Recovery
- Viewing Microsoft Transaction Server In-Doubt Transactions
- Modifying Registry Values for Oracle Fail Safe Configurations

Microsoft Transaction Server Configuration Requirements

You must configure the Microsoft Transaction Server and Oracle Database environments after installing or migrating Oracle Services for Microsoft Transaction Server (OraMTS).

Configuration is not required on the Windows computer if a Microsoft Transaction Server is installed on a computer.

To configure the Microsoft Transaction Server, perform the following tasks on the computer where the Oracle Database is installed:

1. Run the `oramtsadmin.sql` script against the database to create the Microsoft Transaction Server administrative user account (the default username is `mtssys`).
2. Schedule automatic transaction recovery.

   See "Scheduling Automatic Microsoft Transaction Server Transaction Recovery"
3. If you have an Oracle Fail Safe configuration, modify the registry values before or after running the `oramtsadmin.sql` script.

   See "Modifying Registry Values for Oracle Fail Safe Configurations".

Microsoft Transaction Server Transaction Recovery Overview

Distributed transaction capabilities are required to use Microsoft Transaction Server with Oracle database. Microsoft Transaction Server-related Oracle transactions become in-doubt transactions when any of the following fail:

- Microsoft Transaction Server application
- Network
- Microsoft Distributed Transaction Coordinator (MS DTC)

An Oracle MTS Recovery Service resolves in-doubt transactions on the computer that started the failed transaction. An Oracle MTS Recovery Service is automatically

installed with Oracle Services For Microsoft Transaction Server. Only one Oracle MTS Recovery Service can be installed for each computer. A scheduled recovery job on each Microsoft Transaction Server-enabled database permits the Oracle MTS Recovery Service to resolve in-doubt transactions.

The Oracle MTS Recovery Service resolves an in-doubt Microsoft Transaction Server transaction in the following order:

1. The DBMS recovery job detects an in-doubt MTS-related transaction.
2. The DBMS recovery job extracts the recovery service's endpoint address from the XID of the in-doubt transaction and requests the recovery service for the outcome of the MTS/MS DTC transaction.
3. The recovery service requests its MS DTC for transaction outcome.
4. The recovery service reports transaction outcome to the DBMS job process.
5. The DBMS recovery job commits or terminates the in-doubt transaction.

Scheduling Automatic Microsoft Transaction Server Transaction Recovery

OraMTS uses server-based recovery to resolve in-doubt transactions originated by MSDTC. To do this, the OraMTS administrator must be able to access the Windows middle-tier node through `UTL_HTTP`. `oramtsadmin.sql` grants execute privileges on `UTL_HTTP` to the OraMTS administrator, as shown in "Configuring Automatic Transaction Recovery"

> **Note:**
>
> Starting with Oracle version 11*g*, the DBA needs to create an access control list (ACL) as shown in "Creating an Access Control List (ACL)".

Creating an Access Control List (ACL)

For Oracle database version 11*g* and later, the DBA must create an access control list (ACL) that grants the OraMTS administrator the privilege to make out-bound HTTP connections. Example 3-1 demonstrates this:

Example 3-1 Creating an ACL List and Adding OraMTS Administrator to it

```
BEGIN
   -- Create the new ACL, naming it "OraMTSadmin.xml", with a description.
   -- This provides the OraMTS administrative user e.g. MTSADMIN user FOO
   -- the privilege to connect
   DBMS_NETWORK_ACL_ADMIN.CREATE_ACL('OraMTSadmin.xml',
                                     'Allow usage to the UTL network packages',
                                     'FOO', TRUE, 'connect');
   -- Now grant privilege to resolve DNS names to the OraMTS administrative user
   DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE('OraMTSadmin.xml' ,
                                        'FOO', TRUE,'resolve');
  -- Specify which hosts this ACL applies to, in this case we are allowing
  -- access to all hosts. if one knew the list of all Windows middle-tier,
  -- these could be added one by one.
```

```
  DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL('OraMTSadmin.xml','*');
END;
```

Configuring Automatic Transaction Recovery

Automatic transaction recovery is performed by scheduling a database job. A database job for in-doubt transactions must be scheduled for each database participating in Microsoft Transaction Server transactions.

Transaction recovery is configured by running the `oramtsadmin.sql` script, which triggers `utl_oramts.sql` and `prvtoramts.plb` scripts to create the PL/SQL package `utl_oramts`. The database view `oramts_2pc_pending` is also created to show in-doubt transactions related to Microsoft Transaction Server transactions.

The `oramtsadmin.sql` script:

- Creates the Microsoft Transaction Server administrator user account.
- Automatically schedules database jobs for transaction recovery every one minute.

  When the database job is run, it checks for unresolved global transactions in the database that are related to Microsoft Transaction Server. Information in the transaction identifiers (XIDs) of the in-doubt transactions identifies the computer on which the transaction was started. The Oracle MTS Recovery Service on that computer resolves the transaction.
- Schedules post-recovery cleanup every half hour.

Schedule automatic transaction recovery in the database by performing these tasks:

- Setting and Starting Up Database Job-Queue Processes
- Creating and Scheduling Automatic Transaction Recovery

Setting and Starting Up Database Job-Queue Processes

The `JOB_QUEUE_PROCESSES` initialization parameter specifies the maximum number of job slaves started on an instance.

To set and start up job-queue processes:

1. Ensure that you have `SYSDBA` privileges.
2. Go to the computer on which the Oracle Database is installed.
3. Start SQL*Plus:

   ```
   C:\> sqlplus /NOLOG
   ```
4. Connect to the database as `SYSDBA`:

   ```
   SQL> CONNECT / AS SYSDBA
   ```
5. Set the `JOB_QUEUE_PROCESSES` initialization parameter:

   ```
   JOB_QUEUE_PROCESSES = 1
   ```

   The default value for this parameter is `0`. Set this parameter to a value greater than `1` if there are many destinations to which to propagate the messages.
6. Shut down the Oracle Database:

   ```
   SQL> SHUTDOWN
   ```

7. Restart the Oracle Database:

```
SQL> STARTUP
```

8. Exit SQL*Plus:

```
SQL> EXIT
```

Creating and Scheduling Automatic Transaction Recovery

The `oramtsadmin.sql` script creates the Microsoft Transaction Server administrator user account with the default username `mtssys`. The Microsoft Transaction Server transaction recovery jobs run under the administrator user account.

The `oramtsadmin.sql` script runs the `utl_oramts.sql` script to grant the following privileges and roles to the administrator user account:

- `CREATE SESSION` role
- `SELECT_CATALOG_ROLE` role
- `FORCE_ANY_TRANSACTION` privilege
- `DBMS_JOBS` package, on which `EXECUTE` privileges are granted
- `DBMS_TRANSACTION` package, on which `EXECUTE` privileges are granted

To create and schedule automatic transaction recovery:

1. Ensure that you have `SYSDBA` privileges.
2. Log on to the computer where the Oracle Database is installed.
3. Start SQL*Plus:

```
C:\> sqlplus /NOLOG
```

4. Connect to the database as `SYSDBA`:

```
SQL> CONNECT / AS SYSDBA
```

5. Run the `oramtsadmin.sql` script:

```
SQL> @ORACLE_BASE\ORACLE_HOME\oramts\admin\oramtsadmin.sql;
```

   You are prompted for the Microsoft Transaction Server administrator username and password. You can accept the default username of `mtssys` and password of `mtssys`, or change them.

6. If you did change the password in step 5, you can change it using this script:

```
SQL> ALTER USER USERNAME IDENTIFIED BY new_password;
```

   To change the username after completing this task, drop the user, rerun the `oramtsadmin.sql` script, and specify a different username when prompted.

7. Exit SQL*Plus:

```
SQL> EXIT
```

A single PL/SQL package, `utl_oramts`, is created in the Microsoft Transaction Server administrator's schema. `utl_oramts` exposes these public procedures and creates this view:

- utl_oramts.show_indoubt Procedure

- utl_oramts.recover_automatic Procedure
- utl_oramts.forget_RMs Procedure
- oramts_2pc_pending View

utl_oramts.show_indoubt Procedure

Use this procedure to view Microsoft Transaction Server in-doubt transactions in the database. This procedure uses the `dbms_output` package to display results.

Description

This procedure requires `SERVEROUTPUT` set to `ON`.

```
SQL> SET SERVEROUTPUT ON

SQL> EXECUTE utl_oramts.show_indoubt;
```

The following information appears:

```
============================================================
currently indoubt transactions
============================================================
formatid       : 21255235
gtrid              : C2229A505904974D81FB7316B147325900000000
bqual          : 5BAB6A6B55CD294AA20335839110829C0100000000901944700050
local txid     : 142.11.202
tx state           : prepared
protocol       : HTTP
endpoint       : middletier-1@foo.com:2030
============================================================
formatid       : 21255235
gtrid              : 259DF9C8DFC5574F8876F0DF4E15CCAD00000000
bqual          : 2C8DCED5B9816244BA2B73CC013EEB870100000000901944700050
local txid     : 2.18.185
tx state       : prepared
protocol       : HTTP
endpoint       : middletier-2@foo.com:2030
```

utl_oramts.recover_automatic Procedure

This procedure is run by the transaction recovery job. An automatic database job is scheduled for `utl_oramts.recover_automatic`. When the job is run, it checks for unresolved global transactions in the database that are related to Microsoft Transaction Server. Information in the XIDs of the in-doubt transactions identifies the computer on which the transaction started. The Oracle MTS Recovery Service is contacted and resolves the transactions.

utl_oramts.forget_RMs Procedure

Use this procedure to request the transaction manager (MS DTC) to forget resolved transactions. This procedure is run by the post-recovery cleanup job.

oramts_2pc_pending View

The view `oramts_2pc_pending` is created by executing `oramtsadmin.sql`. `oramts_2pc_pending` shows in-doubt transactions in the database. This view consists of the following columns:

Formatid

This is the `formatid` of the global transaction in the database.

global_transaction_id

This is the transaction identifier of the Oracle global transaction corresponding to the Microsoft Transaction Server transaction. In fact, this is the globally unique identifier (GUID) of the Microsoft Transaction Server transaction.

branch_id

This shows the branch identifier of the Oracle transaction. A single Microsoft Transaction Server transaction can have multiple Oracle global transactions. This depends on the number of Microsoft Transaction Server/COM+ components that span the same Microsoft Transaction Server transaction. All these transactions have the small global transaction identifier, but different branch identifiers.

local_tx_id

A local Oracle transaction corresponds to each Microsoft Transaction Server transaction. This column shows the identifier corresponding to this local transaction.

state

This shows the state of the transaction: pending, heuristically committed, heuristically terminated, and so on.

protocol

This is the protocol that the transaction recovery job in the database uses to communicate with the Oracle MTS Recovery Service.

endpoint

This is the endpoint of the Windows computer on which the Microsoft Transaction Server transaction originated. For HTTP connections, this translates to a hostname and port number.

Viewing Microsoft Transaction Server In-Doubt Transactions

To view Microsoft Transaction Server–related in-doubt transactions in the database, use SQL*Plus to query the view `oramts_2pc_pending`:

1. Start SQL*Plus with the Microsoft Transaction Server administrator user account:

   ```
   C:\> sqlplus mtsadmin_user/ mtsadmin_password
   ```

2. Enter the following command:

   ```
   SQL> SELECT * FROM oramts_2pc_pending;
   ```

   This displays the computer on which the in-doubt transaction originated.

Modifying Registry Values for Oracle Fail Safe Configurations

In typical configurations, the MS DTC and Oracle MTS Recovery Service run on the same computer. This ensures that the required information for transaction recovery is available to the Oracle-Microsoft Transaction Server integration layer.

In configurations where the Microsoft Transaction Server application is part of a Windows cluster (for example, the application can fail over to another node or host in the cluster), the MS DTC runs as a cluster-wide resource. All cluster nodes use a single instance of the MS DTC running on any cluster node.

If you have an Oracle Fail Safe configuration, make sure the following registry information is replicated on all nodes in the cluster participating in Microsoft Transaction Server transactions:

To modify registry values for Oracle Fail Safe configurations:

1. Go to the computer on which the MS DTC and Oracle MTS Recovery Service are installed.
2. Start the registry from the command prompt:

   ```
   C:\> regedt32
   ```

   The Registry Editor window appears.
3. Go to `HKEY_LOCAL_MACHINE\Software\Oracle\OracleMTSRecoveryService`.
4. Copy the registry information appearing here to all nodes in the cluster.
5. Reboot the computer on which you added the key.

4

Programming with Microsoft Transaction Server and an Oracle Database

These topics describe how to program with Microsoft Transaction Server and an Oracle Database.

- COM Component Integration in a Transaction
- Microsoft Transaction Server Application Development
- OCI Integration with Microsoft Transaction Server
- ODBC Integration with Microsoft Transaction Server Overview

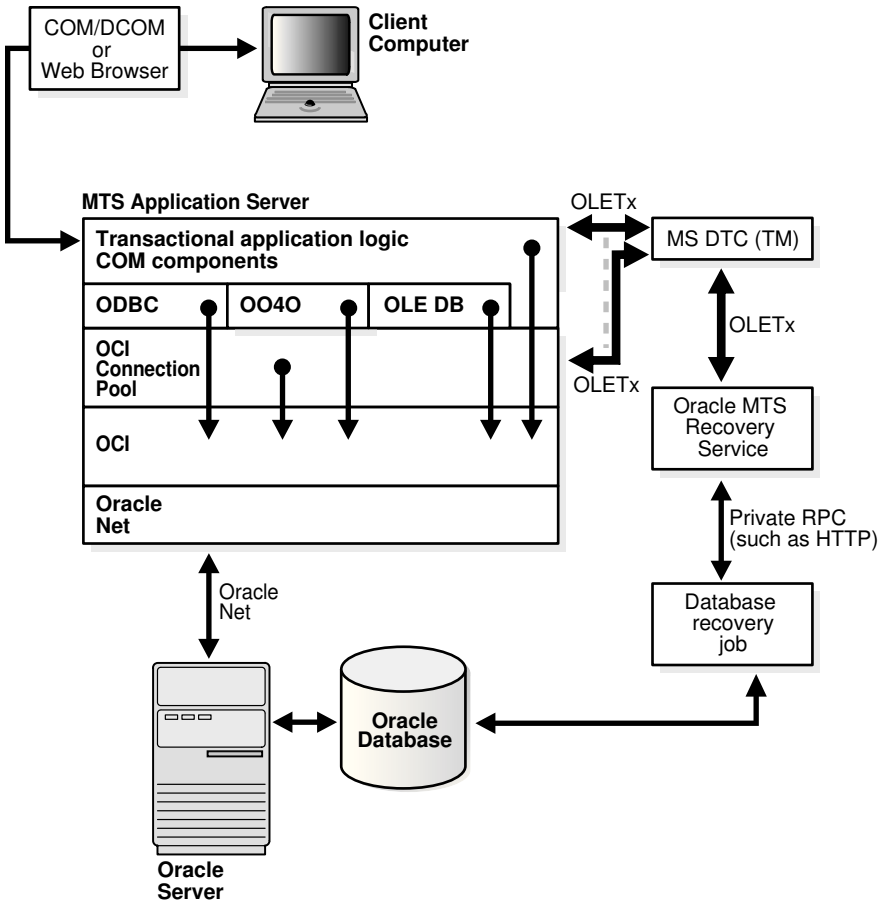
OraMTS also provides integration with Oracle Provider for OLE DB, and Oracle Data Provider for .NET.

> **See Also:**
>
> - *Oracle Provider for OLE DB Developer's Guide for Microsoft Windows* for information on using Oracle Provider for OLE DB with MTS
> - *Oracle Data Provider for .NET Developer's Guide for Microsoft Windows* for information on using Oracle Data Provider for .NET with MTS

COM Component Integration in a Transaction

The focal point of the transaction process is a component of Microsoft Transaction Server called Microsoft Distributed Transaction Coordinator (MS DTC). When a client computer starts a business method on a transactional component, Microsoft Transaction Server begins a transaction coordinated by the MS DTC. The Oracle connection pooling layer enables the database to act as a resource manager (RM) in the MS DTC-coordinated transaction. Figure 4-1 illustrates this transactional model.

Figure 4-1 Component Integration in a Transaction



Client Computer

The client computer activates the application components on the MTS Application Server through a Web browser or through the component object model (COM) / distributed component object model (DCOM).

MTS Application Server

The MTS application server consists of the services that the Windows operating service provides to host transactional application components that a client computer can activate, either indirectly through a Web browser or directly through the component object model (COM) /distributed component object model (DCOM). In response to client requests, the application server invokes the COM components. The invocations are performed within the scope of transactions where required

Transactional Application Logic COM Components

Three primary responsibilities:

- Embed the business logic. If a component is transactional, Microsoft Transaction Server starts a transaction for every method invocation on that component.

- Acquire pooled connections to a Oracle Database through the Oracle resource dispenser and Oracle Call Interface (OCI), Oracle Open Database Connectivity (ODBC) Driver, or Oracle Provider for OLE DB.
- Decide the outcome of the operation by notifying Microsoft Transaction Server of its decision to commit or terminate the changes to all RMs.

Oracle ODBC Driver, Oracle Provider for OLE DB, and OCI

Two primary responsibilities:

- Obtain a service context to the Oracle Database through the OCI connection pooling component.
- Provide connection pooling resources, if necessary (through Oracle Provider for OLE DB or Oracle ODBC Driver). The Oracle ODBC Driver provides pooled ODBC connections. Oracle Provider for OLE DB provides pooled data source objects.

OCI Connection Pool

Three primary responsibilities:

- Enlists the RM (Oracle Database) in the component's Microsoft Transaction Server transaction.
- Starts an Oracle global transaction corresponding to the Microsoft Transaction Server transaction of which the component is a part.
- Acts as a resource dispenser to perform client-side connection pooling.

Oracle Net

Provides connectivity in distributed, heterogeneous computing environments.

Oracle MTS Recovery Service

Recovers in-doubt Oracle transactions that originated from the host computer and are related to the Microsoft Transaction Server.

Database Recovery Job

Detects in-doubt DTC transactions. This job extracts the recovery service's endpoint address in the in-doubt transaction's XID and then requests the outcome of the Microsoft DTC transaction from the recovery service. Ultimately, the job will commit or terminate the in-doubt transaction when it receives the transaction's outcome.

Microsoft DTC

Microsoft Distributed Transaction Coordinator is part of Microsoft Transaction Server and has two primary responsibilities:

- Commits and terminates transactions using the two-phase commit protocol.
- Monitors transactions that require recovery. Multiple MS DTCs can be involved in a single transaction. When a transactional Microsoft Transaction Server component on computer A invokes another transactional Microsoft Transaction Server component on computer B, a connection is opened between the MS DTC on computer A and the MS DTC on computer B. When the root MS DTC commits or terminates a transaction, it sends the request through all involved MS DTCs.

The transaction request is then passed to the OCI connection pooling/Microsoft Transaction Server integration, which sends it to the database.

Oracle Database

Acts as an RM for Microsoft Transaction Server. This is the database on which the client transaction request is performed.

Microsoft Transaction Server Application Development

OCI connection pooling is used to coordinate a transaction in nearly all application programming interfaces. This topic describes how transactions are registered and how OCI connection pooling coordinates them.

Microsoft Transaction Server Component Registration

Application components that run in the Microsoft Transaction Server environment are created as dynamic link libraries (DLLs). Application components are registered with Microsoft Transaction Server using the Microsoft Transaction Server Explorer graphical user interface (GUI) tool.

Types of Registration Components

When you register the application component, you mark it as one of the following types:

- **Requires a Transaction** The component must run in a transaction. If the transaction does not currently exist, Microsoft Transaction Server automatically creates a new transaction for each method invocation on the component.
- **Supports a Transaction** The component can run within the client's transaction. When a new component is created, its context inherits the transaction from the context of the invoking client. If the client does not have a transaction, the new context is also created without one.
- **Requires a New Transaction** The component must run within its own transaction. Microsoft Transaction Server automatically creates a new transaction for each method invocation on the component.
- **Does Not Support Transactions** The component does not run within a transaction. Each method invocation on the component is performed without a surrounding transaction, regardless of whether the invoking client includes a transaction.

Registration of Components

How you register an application component determines if it runs in a Microsoft Transaction Server-coordinated transaction.

- If the application component **runs** in a Microsoft Transaction Server-coordinated transaction, the OCI connection pooling is always used and Microsoft Transaction Server and its MS DTC component coordinate the creation, startup, management, and commitment phases of the transaction. Microsoft Transaction Server ensures that all changes made by the component are committed if the transaction succeeds, or are terminated if the transaction fails.

- If the application component **does not run** in a Microsoft Transaction Server-coordinated transaction, the component runs in a Microsoft Transaction Server environment, but the databases that it accesses may or may not take part in MS DTC-coordinated transactions. If the transaction is not MS DTC-coordinated, the client application must create, start, manage, and commit the transaction. OCI connection pooling may be used, depending upon the interface accessing the database (such as Oracle Provider for OLE DB, Oracle ODBC Driver, or others).

Microsoft Transaction Server-Coordinated Component Transaction

This topic describes how OCI connection pooling, Microsoft Transaction Server, and MS DTC operate with application components in a Microsoft Transaction Server-coordinated transaction environment.

1. The client API (one of Oracle ODBC Driver, OCI, ODP.NET or Oracle Provider for OLE DB) calls OCI function `OraMTSSvcGet()` to obtain a service context from the OCI connection pooling component.
2. The OCI connection pooling component enlists the transaction that will be coordinated by the MS DTC component of Microsoft Transaction Server.

   The OCI service and environment handles are returned to client applications.
3. The client application:
   - Performs the database operations.
   - Calls OCI function `OraMTSSvcRel()` to release the OCI pooling connection obtained at the beginning of the transaction.
   - Calls `SetComplete` (to commit database operations) or `SetAbort` (to terminate database operations) on the Microsoft Transaction Server context object associated with the component.
4. MS DTC performs the two-phase commit protocol to prepare and commit or to terminate the transaction. This notifies the OCI connection pooling component and ends the transaction.
5. OCI connection pooling is notified and performs the necessary steps to complete phase one, the prepare phase, and phase two, the commit or terminate phase.

Microsoft DTC-Coordinated Component Transaction

This topic describes how OCI connection pooling, Microsoft Transaction Server, and MS DTC operate with application components **not running** in a Microsoft Transaction Server-coordinated transaction, but using MS DTC.

1. The client application starts an MS DTC transaction and connects to the Oracle Database. The connection protocol follows one of the following scenarios:
   - Nonpooled OCI connections are obtained through OCI logon calls such as `OCIServerAttach()` and `OCISessionBegin()`. For these connections, the application calls `OraMTSEnlCtxGet()` to associate the OCI service context with a Microsoft Transaction Server enlistment context.
   - A connection pool is obtained by calling `OraMTSSvcGet(..,..,ORAMTS_CFLG_NOIMPLICIT)`.
2. The client handles the context in one of the following scenarios:

- For nonpooled connections, the client application passes in the enlistment context to `OraMTSJoinTxn()`.
- For pooled connections, the client application passes the OCI service context into `OraMTSSvcEnlist()`.

3. The OCI connection pooling component enlists the connection, either pooled or nonpooled, in the transaction coordinated by the MS DTC component of Microsoft Transaction Server.
4. The client application then:
   - Performs database operations.
   - Calls `OraMTSSvcEnlist()` with a `NULL` transaction reference to de-enlist from an MS DTC coordinated transaction.

     For nonpooled connections, `OraMTSTxnJoin()` is invoked with a `NULL` transaction reference to perform the de-enlistment.
   - Calls `OraMTSSvcRel()` to release a pooled connection back to the pool.

     For nonpooled connections, the client calls `OraMTSEnlCtxRel()` to release the enlistment context and then logs off the database.
   - Calls the commit or terminate method on the MS DTC transaction object, such as `pTransaction->Commit()` or `pTransaction->Abort()`.
5. MS DTC performs the two-phase commit protocol to commit the transaction.
6. OCI connection pooling is notified and performs the necessary steps to complete phase one, the prepare phase, and phase two, the commit or terminate phase.

OCI Integration with Microsoft Transaction Server

Example 4-1 illustrates how you can integrate the MTS sever with OCI. The only change in code you must make involves obtaining and releasing the OCI service context handle. Both OCI service context handle and environment handle are acquired when you obtain a pooled OCI connection to the database by calling `OraMTSSvcGet()`. Include the `oramts.h` header and link with the `oramts.lib` library. When you are finished, call OCI function `OraMTSSvcRel()` to release the service context handle and environment handle. Using `OraMTSSvcGet()` enables you to receive connection pooling and implicit transaction support if you registered the application component to run in a Microsoft Transaction Server transaction.

Ensure that for each process, you call `OCIInitialize` at least once before executing any other OCI calls. This initializes the OCI process environment. In addition, you must pass it the `OCI_THREADED` flag. If you are using Microsoft Internet Information Server (IIS) and the components are being called as in-process libraries, then `OCIInitialize` is already called for you. The registry key `ORAMTS_OCI_OBJ_MODE` has been added. Set the value to 1 to initialize OCI in Object mode; otherwise OCI will initialize in the threaded mode.

Example 4-1 Integration of MTS and OCI

```
#include <oci.h>
#include  <oramts.h>
#include  <xolehlp.h>
// other MTS relevant includes ...

// prototype for the error handler.
```

```
BOOL Chekerr(sword swOCIStat, OCIError *OCIErrh);

// MTS component method
HRESULT OCITestMethod()
{
 IObjectContext *pObjectContext = NULL;
 OCIEnv    *myenvh = NULL;
 OCISvcCtx *mysvch = NULL;
 OCIError  *myerrh = NULL;
 OCIStnt   *mystmh = NULL;
 DWORD      dwStat;
 HRESULT    hRes = S_OK;
 sword      swOCIStat;
 BOOL       bCommit = FALSE;
 char      *lpzStmt = "UPDATE EMP SET SAL = SAL + 1000";

 // Initialize the OCI environment first -- request OCI_THREADED
 OCIInitialize(OCI_THREADED, (dvoid*)NULL,NULL,NULL,NULL);
 // attempt to get a connection to the database through the resource dispenser
 OraMTSSvcGet(
"hr","hr_password","finprod_db",&mysvch, &myenvh, ORAMTS_CFLG_ALLDEFAULT);
 // validate return status
 if(dwStat != ORAMTS_ERR_NOERROR)
 {
   printf("error: failed to obtain a connection to the database - %ld",
dwStat);
   goto cleanup;
 }
 // successful logon and enlistment in the MTS transaction. allocate statement
 // handles and other handles using the OCI environment handle myenvh ....
 swOCIStat = OCIHandleAlloc(myenvh, (void *)&myerrh,OCI_HTYPE_ERROR, 0 , NULL);
 if (Checkerr(swOCIStat, myerrh)) goto cleanup;
 swOCIStat = OCIHandleAlloc(myenvh, (dvoid *)&mystmh,OCI_HTYPE_STMT, 0,NULL);
 if (Checkerr(swOCIStat, myerrh)) goto cleanup;
 // prepare a DML statement
 OCIStmtPrepare(mystmh, myerrh, lpzStmt, lstrlen(lpzStmt), OCI_NTV_SYNTAX,
OCI_DEFAULT)
 Checkerr(swOCIStat, myerrh);
 // execute the statement -- ensure that AUTOCOMMIT is not requested.
 OCIStmtExecute(mysvch, mystmh, myerrh, 1, 0, NULL, NULL, OCI_DEFAULT);
 if (Checkerr(swOCIStat, myerrh)) goto cleanup;
 // all's well so far choose to go for a commit
 bCommit = TRUE;
cleanup:
 if (mystmh) OCIHandleFree((void*)mystmh, OCI_HTYPE_STMT);
 if (myerrh  OCIHandleFree((void*)myerrh, OCI_HTYPE_ERROR);
 if (mysvch) OraMTSSvcRel(mysvch);
 if (bCommit)
     pObjectContext->SetComplete();
 else
     pObjectContext->Abort();
 return(bCommit ? S_OK : E_FAIL);
}
```

Integrating COM Components

There are several scenarios for integrating COM components. COM applications that are not hosted by the Microsoft Transaction Server environment, also known as standalone applications, cannot use declarative transactions through the Microsoft

Transaction Server Explorer Microsoft Management Console, but they can use the last three of the scenario described.

COM Components Running in an MTS-Coordinated Transaction

COM components that are running in an MTS-coordinated transactions use OCI connection pooling to implicitly enlist the database in a transaction. The following pseudo-code listing illustrates the use of OCI functions:

```
OCIInitialize(OCI_THREADED, ...)
OraMTSSvcGet(..., &OCISvc, ..., ORAMTS_CFLAG_ALLDEFAULT)
...
OraMTSSvcRel(OCISvc)
```

Non-Transactional COM Components Running with OCI Connection Pooling

COM components that are marked as non-transactional and running in an MTS-coordinated transaction use OCI connection pooling **do not enlist** the database in a transaction. The following pseudo-code listing illustrates the use of OCI functions:

```
OCIInitialize(OCI_THREADED, ...)
OraMTSSvcGet(..., &OCISvc, ..., ORAMTS_CFLAG_NOIMPLICIT)
...
OraMTSSvcRel(OCISvc)
```

COM Components Using MS DTC and OCI Connection Pooling

COM components that are not running in an MTS-coordinated transaction use MS DTC with OCI connection pooling to explicitly enlist the database in a transaction. The following pseudo-code listing illustrates the use of OCI functions:

```
OCIInitialize(OCI_THREADED, ...)
DTCGetTransactionManager(...)
BeginTransaction(..., &transaction)
OraMTSSvcGet(..., &OCISvc, ..., ORAMTS_CFLAG_NOIMPLICIT)
OraMTSSvcEnlist(OCISvc, ..., transaction, ...)
...
OraMTSvcEnlist(OCISvc, ..., NULL, ...)
OraMTSSvcRel(OCISvc)
```

COM Components Using MS DTC and Nonpooling OCI Connection

COM components that are not running in an MTS-coordinated transaction use MS DTC with a non-pooling OCI connection to explicitly enlist the database in a transaction. The following pseudo-code listing illustrates the use of OCI functions:

```
OCIInitialize(OCI_THREADED, ...)
OCI to get connected
OraMTSEnlCtxGET
DTCGetTransactionManager(...)
BeginTransaction(..., &transaction)
OraMTSJoinTxn (OCISvc, ..., transaction, ...)
...
OraMTSJoinTxn
...
OraMTSEnlCtxRel()
OCI to logoff
```

Using OCI Functions

This topic details the OCI functions discussed earlier in this topic. Table 4-1 summarizes these functions.

Table 4-1 Summary of OCI Functions for Integrating MTS and Oracle Database

OCI Function	Summary
OraMTSSvcGet()	Obtains a pooled connection from the OCI connection pool.
OraMTSSvcRel()	Releases a pooled OCI connection, OCI service context, back to the connection pool.
OraMTSSvcEnlist()	Enlists or de-enlists an OCI connection in a transaction coordinated by MS DTC.
OraMTSSvcEnlistEx()	Enlists an OCI connection or service context in an MS DTC transaction.
OraMTSEnlCtxGet()	Creates an enlistment context for a nonpooled OCI connection.
OraMTSEnlCtxRel()	Eliminates a previously set up enlistment context for a nonpooled OCI connection.
OraMTSJoinTxn()	Enlists a nonpooled OCI connection in an MS DTC transaction.
OraMTSTransTest()	Tests if you are running inside a Microsoft Transaction Server-started transaction.
OraMTSOCIErrGet()	Retrieves the OCI error code and message text.

OraMTSSvcGet()

Obtains a pooled connection, also known as an OCI service context, from the OCI connection pool. The pooled connection includes an OCI service context handle and an OCI environment handle.

Syntax

```
DWORD  OraMTSSvcGet(

                text*       lpUname,
                text*       lpPsswd,
                text*       lpDbnam,
                OCISvcCtx** pOCISvc,
                OCIEnv**    pOCIEnv,
                ub4         dwConFlgs);
```

Parameters

Table 4-2 OraMTSSvcGet() Parameters

Parameter	IN/OUT	Description
lpUname	IN	Username for connecting to the Oracle Database
lpPsswd	IN	Password for the username

Table 4-2 (Cont.) OraMTSSvcGet() Parameters

Parameter	IN/OUT	Description
lpDbnam	IN	The net service name for connecting to the database (created with Oracle Net Manager or Oracle Net Configuration Assistant)
pOCISvc	OUT	Pointer to the OCI service context handle
pOCIEnv	OUT	Pointer to the OCI environment handle
dwConFlgs	IN	Connection flags. Possible values are: • ORAMTS_CFLG_ALLDEFAULT Obtains a pooled connection and enlists the connection in any Microsoft Transaction Server transaction, if one exists. If the component is nontransactional, no enlistment request is dispensed. • ORAMTS_CFLG_NOIMPLICIT Obtains a pooled connection, but does not enlist the resource in any Microsoft Transaction Server transaction even if the component is transactional. Use this flag if the component enlists the connection resource later using OraMTSSvcEnlist(). Prior to releasing a connection obtained in this fashion, the client must de-enlist the resource if enlisted. • ORAMTS_CFLG_UNIQUESRVR Requests a single OCI session for each OCI Server. In this release, multiplexing is not supported. Therefore, this option is always used. • ORAMTS_CFLG_SYSDBALOGN Use this flag if connecting as SYSDBA. • ORAMTS_CFLG_SYSOPRLOGN Use this flag if connecting as SYSOPER. • ORAMTS_CFLG_PRELIMAUTH Use this flag if connecting as the user INTERNAL to pre-Oracle9*i* databases. The INTERNAL account is no longer valid as of Oracle9*i*. Instead, log on with a SYSDBA or SYSOPER account using the ORAMTS_CFLG_SYSOPRLOGN or ORAMTS_CFLG_SYSDBALOGN flag.

Returns

Returns ORAMTSERR_NOERROR upon successful acquisition of an OCI pooling connection (OCI service context).

Usage Notes

- OraMTSSvcGet() returns a pooled OCI connection to the caller, enabling a database transaction using OCI to begin. Use OraMTSSvcGet() to implicitly enlist the OCI connection in a transaction coordinated by Microsoft Transaction Server. In this type of transaction, Microsoft Transaction Server controls the creation, startup, management, and commitment phases of the transaction through its MS DTC component.

- `OraMTSSvcGet()` also provides connection pooling without enlisting the Oracle Database in a Microsoft Transaction Server transaction. This is done by setting `OraMTSSvcGet()` as follows:

  ```
  OraMTSSvcGet(...,ORAMTS_CFLG_NOIMPLICIT)
  ```

- In all cases where `OraMTSSvcGet()` is used, you must always use `OraMTSSvcRel()` to release the connection when finished.
- Use the flags `ORAMTS_CFLG_SYSDBALOGN` and `ORAMTS_CFLG_SYSOPRLOGN` when connecting as `SYSDBA` and `SYSOPER`, respectively.
- To obtain a nonenlisted connection using the `hr/`*`hr_password`* account, call `OraMTSSvcGet()` as follows:

  ```
  OraMTSSvcGet("hr", "hr_password", "oracle", &OCISvc, &OCIEnv,
  ORAMTS_CFLG_ALLDEFAULT | ORAMTS_CFLG_NOIMPLICIT);
  ```

- `OraMTSSvcGet()` does not support placing the username (`lpUname`), password (`lpPsswd`), and net service name syntax (`lpDbname`) together in the username argument (for example, `hr/`*`hr_password`*`@prod_fin`). Instead, the caller must fill in `lpUname`, `lpPsswd`, and `lpDbname` separately (as shown in the previous syntax example). Calling `OraMTSSvcGet()` with the username and password as `NULL` strings uses external authentication (operating system authentication) for the connection.

OraMTSSvcRel()

Releases a pooled OCI connection, OCI service context, back to the connection pool. Use this function to release connections that were acquired with `OraMTSSvcGet()`.

Syntax

```
DWORD OraMTSSvcRel(OCISvcCtx* OCISvc);
```

Parameters

Table 4-3 OraMTSSvcRel() Parameters

Parameter	IN/OUT	Description
`OCISvc`	`IN`	OCI service context for a pooled connection

Returns

Returns `ORAMTSERR_NOERROR` upon successful release of a pooled OCI connection.

Usage Notes

- An OCI pooled connection obtained through a previous call to `OraMTSSvcGet()` is released back to the connection pool. Once released back to the connection pool, the OCI service context, its environment handle, and all child handles are invalid.
- A nontransactional client component must explicitly call `OCITransCommit()` or `OCITransAbort()` prior to releasing a connection obtained through `OraMTSSvcGet(..., ...,ORAMTS_CFLG_ALLDEFAULT)` back to the pool. Otherwise, all changes made in that session are rolled back. A transaction component uses

the SetComplete or SetAbort methods on its Microsoft Transaction Server object context.

- Components that have called OraMTSSvcGet(..., ...,ORAMTS_CFLG_NOIMPLICIT) to obtain a connection resource must first de-enlist the resource if enlisted. If the connection was enlisted explicitly, pTransaction->Commit() or pTransaction->Abort() must be called. Otherwise, OCITransCommit() or OCITransAbort() must be called before releasing the connection back to the pool.

OraMTSSvcEnlist()

Enlists or de-enlists an OCI connection in a transaction coordinated by MS DTC. Use this call to explicitly enlist pooled connections. Nonpooled connections must enlist with OraMTSJoinTxn().

Syntax

```
DWORD OraMTSSvcEnlist(

                  OCISvcCtx*  OCISvc,
                  OCIError*   OCIErr,
                  void*       lpTrans,
                  unsigned    dwFlags);
```

Parameters

Table 4-4 OraMTSSvcEnlist() Parameters

Parameter	IN/OUT	Description
OCISvc	IN	OCI service context for pooled connections obtained by calling OraMTSSvcGet()
OCIErr	IN/OUT	OCI error handle (ignored)
lpTrans	IN	Pointer to the MS DTC-controlled transaction in which to enlist. If NULL, the OCI connection is de-enlisted from the MS DTC-controlled transaction.
dwFlags	IN	Flag used for enlisting in a transaction. Use the ORAMTS_ENFLG_DEFAULT value. If enlisting, then start a new Oracle global transaction. If de-enlisting, then detach from any global Oracle transaction and delete the context object if the OCI service context represents a nonpooled connection.

Returns

Returns ORAMTSERR_NOERROR on success.

Usage Notes

- Use this call to explicitly enlist or de-enlist a pooled connection. For enlisting and de-enlisting nonpooled connections, use OraMTSSvcRel().
- OraMTSSvcEnlist() enlists (or de-enlists) pooled OCI connections obtained previously through OraMTSSvcGet() with the ORAMTS_CFLG_NOIMPLICIT flag and not yet released with OraMTSSvcRel(). The pooled OCI connections must be explicitly enlistable. When the transaction is complete, you must de-enlist OraMTSSvcEnlist(), passing NULL as the transaction pointer as follows:

```
OraMTSSvcEnlist (OCISvc, OCIErr, NULL, ORAMTS_ENFLG_DEFAULT)
```

You must use `OraMTSSvcRel()` to release the connection when done.

- Callers must allocate a connection, enlist the connection, perform work, de-enlist the connection, release the connection, and then attempt to commit or terminate.

OraMTSSvcEnlistEx()

Enlists an OCI connection or service context in an MS DTC transaction. Use this call only to explicitly enlist pooled connections. Nonpooled connections must enlist with `OraMTSJoinTxn()`.

Syntax

```
DWORD OraMTSSvcEnlistEx(
                    OCISvcCtx* OCISvc,

                    OCIError*  OCIErr,
                    void*      lpTrans,
                    unsigned   dwFlags,
                    char*      lpDBName);
```

Parameters

Table 4-5 OraMTSSvcEnlistEx() Parameters

Parameter	IN/OUT	Description
`OCISvc`	`IN`	OCI service context for pooled connections obtained by calling `OraMTSSvcGet()`
`OCIErr`	`IN/OUT`	OCI error handle (ignored)
`lpTrans`	`IN`	Pointer to the MS DTC-controlled transaction in which to enlist. If `NULL`, the OCI connection is de-enlisted from the MS DTC-controlled transaction.
`dwFlags`	`IN`	Flag used for enlisting in a transaction. Use the `ORAMTS_ENFLG_DEFAULT` value. If enlisting, then start a new Oracle global transaction. If de-enlisting, then detach from any global Oracle transaction and delete the context object if the OCI service context represents a nonpooled connection.
`lpDBName`	-	Net service name for connecting to the database (created with Oracle Net Manager or Oracle Net Configuration Assistant)

Returns

Returns `ORAMTSERR_ILLEGAL_OPER`.

Usage Notes

Use `OraMTSSvcEnlistEx()` for pooled connections or `OraMTSJoinTxn()` for nonpooled connections.

OraMTSEnlCtxGet()

Creates an enlistment context for a nonpooled OCI connection.

Syntax

```
DWORD OraMTSEnlCtxGet(
                    text*       lpUname,
                    text*       lpPsswd,
                    text*       lpDbnam,
                    OCISvcCtx*  pOCISvc,
                    OCIError*   pOCIErr,
                    ub4         dwFlags,
                    void**      pCtxt);
```

Parameters

Table 4-6 OraMTSEnlCtxGet() Parameters

Parameter	IN/OUT	Description
lpUname	IN	Username for connecting to the Oracle Database
lpPsswd	IN	Password for connecting to the Oracle Database
lpDbnam	IN	Net service name for connecting to a database
pOCISvc	IN	OCI service context for a nonpooled connection
pOCIErr	IN	OCI error handle
dwFlags	IN	Enlistment flags. The only value currently permitted is 0.
pCtxt	OUT	Enlistment context to be created

Returns

Returns ORAMTSERR_NOERROR on success.

Usage Notes

- This call sets up an enlistment context for a nonpooled connection. This call must be started just after the caller establishes the OCI connection to the database. Once created, this context can be passed into OraMTSJoinTxn() calls. Prior to deleting the OCI connection, OraMTSEnlCtxRel() must be called to delete the enlistment context.
- Callers must:
  - Allocate a nonpooled connection through OCI.
  - Create an enlistment context by calling OraMTSEnlCtxGet().
  - Enlist the connection by calling OraMTSJoinTxn().
  - Perform database work.
  - De-enlist the connection by calling OraMTSJoinTxn() with a NULL transaction pointer.
  - Attempt to commit or terminate work.
  - Release the enlistment context by calling OraMTSEnlCtxRel().
  - Release the nonpooled OCI connection and delete its associated OCI environment handle.

OraMTSEnlCtxRel()

Eliminates a previously set up enlistment context for a nonpooled OCI connection.

Syntax

```
DWORD OraMTSEnlCtxRel(void* pCtxt);
```

Parameters

Table 4-7 OraMTSEnlCtxRel() Parameters

Parameter	IN/OUT	Description
pCtxt	IN	Enlistment context to eliminate

Returns

Returns ORAMTSERR_NOERROR on success.

Usage Notes

- Before dropping a nonpooled OCI connection, a client must call OraMTSEnlCtxRel() to eliminate any enlistment context it may have created for that connection. The enlistment context can maintain OCI handles allocated off the connection's OCI environment handle. This makes it imperative that the environment handle is not deleted for the associated enlistment context.

OraMTSJoinTxn()

Enlists a nonpooled OCI connection in an MS DTC transaction.

Syntax

```
DWORD OraMTSJoinTxn(void*  pCtxt,
                    void*  pTrans);
```

Parameters

Table 4-8 OraMTSJoinTxn() Parameters

Parameter	IN	Description
pCtxt	IN	Enlistment context for the OCI connection
pTrans	IN	Reference to the MS DTC transaction object

Returns

Returns ORAMTSERR_NOERROR on success.

Usage Notes

- Clients use this call with nonpooled OCI connections to enlist connections in MS DTC-coordinated transactions. The client passes in the wide reference to the enlistment context representing the OCI connection, along with a reference to an MS DTC transaction object. If `pTrans` is `NULL`, the OCI connection is de-enlisted from any MS DTC transaction in which it is currently enlisted. You can enlist a previously-enlisted OCI connection in a different MS DTC transaction.

OraMTSTransTest()

Tests if you are running inside a Microsoft Transaction Server-started transaction.

Syntax

```
BOOL OraMTSTransTest();
```

Returns

Returns `true` if running inside a Microsoft Transaction Server transaction.

Usage Notes

Microsoft Transaction Server transactional components use `OraMTSTransTest()` to check if a component is running within the context of a Microsoft Transaction Server transaction. Note that this call can only test Microsoft Transaction Server-started transactions. Transactions started by directly calling the MS DTC are not detected.

OraMTSOCIErrGet()

Retrieves the OCI error code and message text, if any, from the last `OraMTS` function operation, typically `OraMTSSvcGet()` or `OraMTSJoinTxn()`.

Syntax

```
BOOL OraMTSOCIErrGet(DWORD* dwErr,
                     LPCHAR lpcEMsg,
                     DWORD* lpdLen);
```

Parameters

Table 4-9 OraMTSOCIErrGet() Parameters

Parameter	IN/OUT	Description
`dwErr`	-	Error code
`lpcEMsg`	-	Buffer for the error message, if any
`lpdLen`	-	Set to the actual number of message bytes

Returns

Returns `true` if an OCI error is encountered. Otherwise, `false` is returned. If `true` is returned and `lpcEMsg` and `lpdLen` are valid, and there is a stashed error message, up

to lpdLen bytes are copied into lpcEMsg. lpdLen is set to the actual number of message bytes.

Usage Notes

Example 4-2 illustrates how OraMTSOCIErrGet() retrieves the OCI error code and OCI error message text, if any, from the last OraMTSSvc() operation on this thread.

Example 4-2 Retrieving the OCI Error Code and Message Text

```
DWORD dwStat = OraMTSSvcGet("hr",
                            "invalid_password",
                            "fin_prod",
                            "db",
                            &mysvch,
                            &myenvh,
                            ORAMTS_CFLG_ALLDEFAULT);

if (dwStat != ORAMTS_ERR_NOERROR)
   {
      DWORD   dwOCIErr;
      char    errBuf[MAX_PATH];
      DWORD   errBufLen = sizeof(effBuf);

      if (OraMTSOCIErrGet(&dwOCIErr, &errBuf, &errBufLen))
            printf("OCIError %d: %s"\n);
   }
```

ODBC Integration with Microsoft Transaction Server Overview

This topic describes how to use Oracle ODBC Driver with Microsoft Transaction Server and a Oracle Database. No changes to OCI code are necessary for ODBC to operate successfully.

Setting the Connection Attribute

To use Microsoft Transaction Server with either Oracle ODBC Driver 11.1 or Microsoft Oracle ODBC driver, set the connection attribute using the SQLSetConnectAttr function to call the parameter SQL_ATTR_ENLIST_IN_DTC in the ODBC code. This enables you to receive connection pooling and implicit transaction support.

Using Oracle ODBC Driver

The ODBC Driver Manager distributed with ODBC 3.0 is a Resource Dispenser that supports connection pooling. Oracle ODBC Driver release 11.1 integrates with the ODBC 3.0 Driver Manager by supporting the SQLSetConnectAttr(...,..., SQL_ATTR_ENLIST_IN_DTC) call to enlist or de-enlist the ODBC connection used in MS DTC-coordinated transactions.

Use the Oracle ODBC Driver 11.1 with:

- Applications you develop
- The sample banking application that Microsoft provides with Microsoft Transaction Server.

To configure Oracle ODBC Driver, follow these steps:

1. Choose **Start** > **Settings** > **Control Panel**.

   The Control Panel window appears.

2. Double-click **ODBC**.

   The ODBC Data Source Administrator dialog box appears.

3. Choose the **File DSN** tab.

4. To make Oracle ODBC Driver work with Microsoft sample banking application demo, follow these steps. Otherwise, skip this step.

   - Back up Microsoft `mtssamples.dsn` file. This file is located in *`ROOTDRIVE`*`:\program files\common files\odbc\data sources`.
   - Select `mtssamples.dsn` and click **Remove**.
   - Click **Yes** when prompted.

     This deletes the configuration file that enables the Microsoft Transaction Server sample application demo to use the Microsoft ODBC driver.

   If you don't intend to use the demo, click **Add to create a new File data source name (DSN)**.

   The Create New Data Source wizard appears.

5. Select **Oracle** in *`HOME_NAME`*.

6. Click **Advanced**.

7. Add the following information in the keywords and values field:

   ```
   SERVER=database_alias
   USERNAME=hr
   PASSWORD=hr_password
   ```

   where:

   - `SERVER` is the The database alias used by the demo to access the database `mtsdemo`.
   - `USERNAME` is the database username for this application, such as `hr`.
   - `PASSWORD` is the database password for username `hr`.

   Verify that the `hr` schema contains the `account` and `receipt` tables.

8. Click **OK**.

9. Click **Next** to continue with the Create New Data Source wizard.

10. For the Microsoft sample application, enter `mtssamples.dsn` (Microsoft ODBC name). This name must exactly match the name you removed in Step 4.

    For applications you develop, enter the name of the DSN file that will be used.

11. Complete the remaining Create New Data Source wizard pages.

12. Click **OK** to exit the ODBC Data Source Administrator dialog box.

13. Exit the Control Panel window.

> **See Also:**
>
> Microsoft Transaction Server SDK for information

Using Microsoft Oracle ODBC Driver

As an alternative to the Oracle ODBC driver, you can use the Microsoft Oracle ODBC Driver. You should be aware that you would not be able to integrate with Oracle Provider for OLE DB and Oracle Data Provider for .NET if using the Microsoft driver. Also, you will not receive the performance benefits of the Oracle ODBC driver, API support for integration, or Oracle client support services.

After enabling the Microsoft Oracle ODBC Driver, perform these additional steps to configure the Microsoft Oracle ODBC Driver:

To configure the Microsoft Oracle ODBC Driver:

1. Install Oracle Required Support Files (RSF) and SQL*Net 2.3 or later on the computer where the Microsoft Oracle ODBC Driver is operating.
2. Run the *ORACLE_BASE**ORACLE_HOME*\oramts\samples\ sql\omtssamp.sql script.
3. Use SQL*Net Easy Config to set up a database alias connection. This alias is used in the mtssamples.dsn file.
4. If you installed the RSFs in a home with Oracle Net installed, be sure to set the following registry parameter at HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE:

   ```
   ORAOCI = ORA73.DLL
   ```

> **See Also:**
>
> "Setting Up MTS to Access Oracle" in the Microsoft Transaction Server online Help for instructions on enabling the Microsoft Oracle ODBC Driver

5

Tuning Microsoft Transaction Server Performance

These topics provide Microsoft Transaction Server performance tuning information.

- Improving Microsoft Transaction Server Application Performance
- Managing Microsoft Transaction Server Connections
- Increasing the Transaction Timeout Parameter
- Changing Initialization Parameter Settings
- Additional Parameters
- Starting MSDTC

Improving Microsoft Transaction Server Application Performance

Optimizing the programming methods of your application improves its performance. For example, placing all code for a given transaction into one component object model (COM) component means you do not mark that component as transactional. This eliminates the overhead of going through Microsoft Transaction Server. You can subsequently use the Oracle commit or rollback functions to control that transaction in the component. If you are using the Oracle Call Interface (OCI), you can still use `ORAMTSSvcGet()`, but you can also use the `ORAMTS_CFLG_NOIMPLICIT` flag. If you are updating across two or more Oracle Database instances, use database links and connect to one database from the COM component.

> **See Also:**
>
> "OCI Integration with Microsoft Transaction Server" for more information on using `ORAMTSSvcGet()`

Managing Microsoft Transaction Server Connections

When a .NET or COM component ends a session with the Oracle Database, the connection does not immediately terminate. Instead, it remains idle in a connection pool, where it is available for reuse by another component attempting a new connection to the Oracle Database.

Connection Pooling Registry Parameters

The idle period during which a connection is reusable reduces the resource costs associated with opening a new connection. The amount of time that the connection remains idle and available in the connection pool is determined by several registry parameter settings. You can modify these parameters on the computers on which the client Microsoft Transaction Server components are installed, in the file `HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\HOMEID`:

ORAMTS_CONN_POOL_TIMEOUT

The time, in seconds, that the connection remains idle and available for reuse in the client side connection pool, before timing out and being released. The default value of this parameter is `120` seconds.

ORAMTS_SESS_TXNTIMETOLIVE

The time, in seconds, that the connection established using `OraMTSSvcGet()` remains alive in the client side connection pool after being released by an `OraMTSSvcRel()` call. The sum of the `ORAMTS_CONN_POOL_TIMEOUT` and `ORAMTS_NET_CACHE_TIMEOUT` values determines the actual time before a connection terminates completely. The default value of this parameter is `120` seconds.

ORAMTS_NET_CACHE_TIMEOUT

The resource dispenser implemented inside Oracle Services for Microsoft Transaction Server establishes pooled connections to Oracle databases. When these connections are no longer in use, the user sessions are disconnected after the timeout specified by `ORAMTS_CONN_POOL_TIMEOUT`. However, the underlying Oracle Net connections are cached for the period specified by this parameter. After this time, expressed in milliseconds, a cached Oracle Net connection to the database will be terminated. The default value of this parameter is 120000 milliseconds. Oracle recommends setting this parameter to a higher value than the value for `ORAMTS_CONN_POOL_TIMEOUT`. The sum of the time periods specified for `ORAMTS_CONN_POOL_TIMEOUT` and `ORAMTS_NET_CACHE_TIMEOUT` determines the actual time before a connection terminates completely.

ORAMTS_NET_CACHE_MAXFREE

The maximum number of free server connections that should be maintained in the client-side connection pool at a given time. The default value of this parameter is `5`.

ORAMTS_OSCREDS_MATCH_LEVEL

The level of Windows security checking implemented when the `OS_ROLES` initialization parameter in the `init.ora` file is `true`.

When a user establishes a connection to the Oracle Database using the `CONNECT` command, the Windows username is associated with specific database roles and privileges. When the user disconnects, this connection becomes idle and available in the pool. When another user enters the `CONNECT` command, if the Windows username is identical to the one used by the first user, the second user can receive the same database roles and privileges as the first user. This is a considerable security concern,

especially if the second user possesses only the CREATE SESSION and RESOURCE database roles but receives the DBA privileges of the first user.

By default, the ORAMTS_OSCREDS_MATCH_LEVEL parameter value is OS_AUTH_LOGIN, and Windows security checking is performed only if the username and password are NULL.

The most secure setting for this parameter is ALWAYS, which ensures that Windows security checking is performed in all cases, and takes care of possible security breaches due to identical non-null Window usernames.

Because Windows security checking is a resource-intensive operation, you may wish to set the value of this parameter to NEVER. However, if you know that OS_ROLES is true, or if you use operating system-authenticated connections, you should avoid this option.

Increasing the Transaction Timeout Parameter

If transaction requests are timing out before completing, the transaction timeout parameter may be set too low. Increase the transaction timeout parameter to ensure that transactions have enough time to complete.

To increase the transaction timeout parameter:

1. Go to the Windows computer on which Microsoft Transaction Server is installed.
2. From the **Start** menu, select **Programs**, then **Administrative Tools**, then **Component Services**.

   The Component Services window appears.
3. Double-click **Console Root** in the **Component Services** window so its tree structure expands.
4. Double-click **Component Services**.
5. Double-click **Computers**.
6. Right-click **My Computer**.

   A menu appears with several options.
7. Choose **Properties**.

   The My Computer Properties dialog box appears.
8. Choose the **Options** tab.
9. Enter a value in the **Transaction Timeout** field and click **OK**.

   The transaction timeout value is increased. For most environments, 60 seconds may be enough. However, if the transaction is competing with numerous concurrent transactions, this value may be too low.

Changing Initialization Parameter Settings

You may need to modify several initialization parameters to use the Oracle Database with Microsoft Transaction Server. The values you should set these parameters to are based on the database workload environment.

To verify initialization parameter file values, follow these steps:

1. Ensure that you have SYSDBA privileges.

2. Go to the computer on which the Oracle Database is installed.
3. Start SQL*Plus:

   ```
   C:\> sqlplus /NOLOG
   ```

4. Connect to the database as SYSDBA:

   ```
   SQL> CONNECT / AS SYSDBA
   ```

5. Check the value for the SESSIONS parameter:

   ```
   SQL> SHOW PARAMETER SESSIONS
   ```

6. Check the value for the PROCESSES parameter:

   ```
   SQL> SHOW PARAMETER PROCESSES
   ```

   The current settings for both SESSIONS and PROCESS parameters are typically appropriate for running the Microsoft application demo. For creating and deploying .NET or COM-based applications, the values for these parameters depend on the database environment's anticipated workload. For example, if you anticipate 100 concurrent connections to the Oracle Database, consider setting both values to 200 to accommodate a possible system overload. Ensure that you do not set these parameters too high, because they are resource-intensive.

7. Set the following initialization parameters to at least these values:
   - SESSIONS = 200 (or larger if anticipating heavier loads)
   - PROCESSES = 200 (or larger if anticipating heavier loads)
8. Shut down the Oracle Database:

   ```
   SQL> SHUTDOWN
   ```

9. Restart the Oracle Database:

   ```
   SQL> STARTUP
   ```

10. Exit SQL*Plus:

    ```
    SQL> EXIT
    ```

> **See Also:**
>
> *Oracle Database Reference* for information about these parameters.

Additional Parameters

Use the registry variable ORAMTS_ABORT_MODE to control whether a new connection always performs an abort or whether the originally enlisted connection can be used to perform the abort, that is, whether the abort is synchronous or asynchronous.

By default, the originally enlisted connection performs transaction aborts (whenever possible).

Registry variable: ORAMTS_ABORT_MODE

Values:

- `ORAMTS_ABORT_MODE_NEW_CONN_ONLY`: Results in asynchronous aborts. A new connection to the database is opened for performing transaction aborts.
- Any other value implies the default behavior.

Starting MSDTC

The Microsoft Distributed Transaction Coordinator (MS DTC) must be running to enable communication with Oracle Services for Microsoft Transaction Server.

To start MS DTC, follow these steps:

1. On the computer where Microsoft Transaction Server is installed, from the **Start** menu, choose **Programs**, then **Administrative Tools**, then **Component Services**.

   The Component Services window appears.
2. In the Component Services Window, expand **Component Services** under the Console Root.
3. Expand **Computers** under Component Services.
4. Right-click My Computer.

   A menu with several options appears.
5. Choose **Start MSDTC**.

   MS DTC starts.

6

Troubleshooting Oracle Microsoft Transaction Server

These topics provide information on troubleshooting Oracle Microsoft Transaction Server.

- Tracking Oracle Services for Microsoft Transaction Server Performance
- Correcting Oracle Net Changes that Impact Connection Pooling
- Designing an Application that Uses Multiple Databases
- Working with Different Types of Connection Pooling
- Working with In-Doubt Transactions
- Dropping the Microsoft Transaction Server Administrative User Account

Tracking Oracle Services for Microsoft Transaction Server Performance

Trace files record information about Oracle Services for Microsoft Transaction Server performance. This information includes:

- Any errors
- Enlistment requests and outcomes
- Prepare, commit, and terminate requests and their outcomes

Registry parameters handle tracing within `oramts.dll`, which performs the following tasks:

- It implements the API for integrating the Oracle Database with Microsoft Transaction Server.
- It works as a resource dispenser to provide pooled Oracle Call Interface (OCI) connections.
- It enables clients with nonpooled OCI connections to enlist in transactions started by Microsoft Distributed Transaction Coordinator (MS DTC).
- It communicates with Oracle Services for Microsoft Transaction Server to enlist the Oracle Database in transactions started by MS DTC.

The MTS-based COM components can acquire connections to both dedicated and shared Oracle servers of a database. The components can then attempt to perform distributed updates, using data manipulation language, on another database using pre-existing database links between these databases. While the distributed updates from shared servers succeed, those from dedicated servers fail.

Registry parameters that handle tracing are automatically set in `\HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\HOMEID` during the installation of Oracle Services for Microsoft Transaction Server.

Restart all applications using Oracle Services for Microsoft Transaction Server. Further, if you have modified parameters related to the tracing of the `OracleMTSRecoveryService`, restart the Windows service.

Table 6-1 shows the range of `ORAMTS_CP_TRACE_LEVEL` trace values.

Table 6-1 ORAMTS_CP_TRACE_LEVEL Trace Registry Parameter Values

Level	Description
0	Disables tracing. If the registry parameter is not set in the registry or as an environment variable, then tracing is disabled. This is equivalent to setting the level to `0`.
1	Traces errors only
2	Traces important events in addition to errors
3	This level is not supported; if you set this parameter to 3, level 2 tracing is enabled.
4	Traces function entry/exit, important events, and errors
5	Traces reference counting function and constructor/destructor entry/exit

> **Note:**
>
> The Oracle MTS Recovery Service also generates trace file output in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\oramts\trace` directory.

Correcting Oracle Net Changes that Impact Connection Pooling

The connection pool provided by the OraMTS layer, `oramts.dll`, uses a connection's net service name to identify pooled connections for an application. If changes are made to the net service name, and pooled connections are available, the application using the connection pool must be stopped and restarted. These changes can include altering the host or the database system identifier (SID) for the net service name in the `tnsnames.ora` file.

These changes ensure that all currently pooled connections corresponding to the old net service name are destroyed and any new pooled connections use the changes made to the net service name. This includes any application hosting Microsoft Transaction Server components.

To empty connection pools, perform the following:

- If the application is an *out-of-process* Microsoft Transaction Server component (*server package*), run the following application:

  ```
  C:\> mtxstop
  ```

  This empties the connection pools.

- If the application is an *in-process* Microsoft Transaction Server component (*library package*), terminate the application. This also empties the connection pool.

Designing an Application that Uses Multiple Databases

Oracle clients can establish connections to a database in two ways:

- Typical Oracle clients establish connections to a database using a dedicated server configuration. In a dedicated server configuration, one client corresponds to one Oracle server process.
- For scalability under heavy loads, Oracle clients have the option of using a shared server configuration. In a shared server configuration, a single Oracle server process can be shared by more than one client connection.

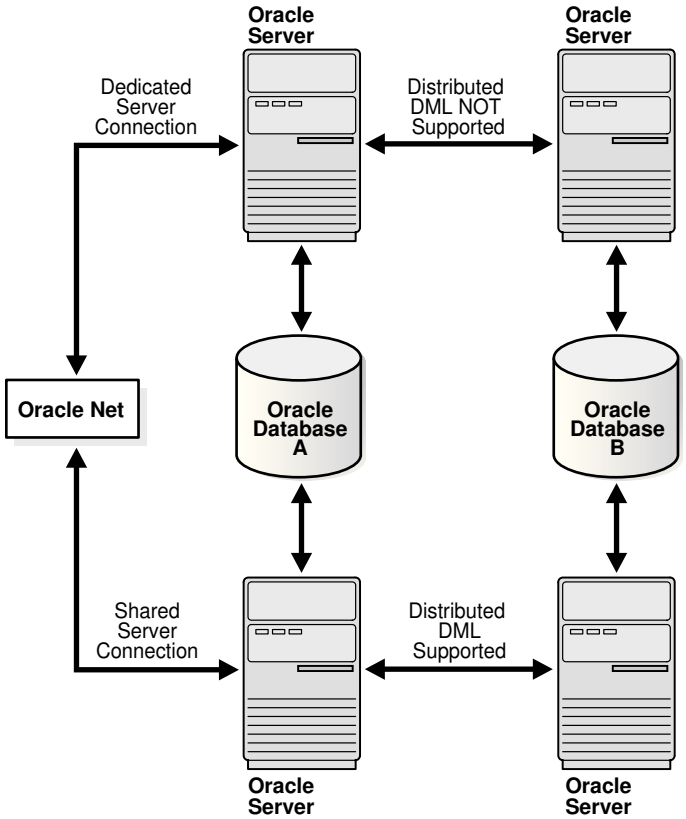
Microsoft Transaction Server communicates with the database through distributed transactions. In a dedicated server configuration, you cannot use distributed updates (data manipulation language statements across database links) from other databases. However, if the original connection to the database is established using shared server configurations, the distributed updates from other databases succeed.

To use data manipulation language statements in shared server configurations, set the following parameter in the `tnsnames.ora` file:

```
SERVER=dedicated
```

This forces the Oracle Net listener to provide a dedicated connection. Figure 6-1 shows this process.

Figure 6-1 Distributed DML Statements from MTS Applications



Working with Different Types of Connection Pooling

Oracle Net Connection Pooling

Oracle Net connection pooling is a server-side feature that is implemented only if the Oracle Database is configured for shared server support. Oracle Net connection pooling enables you to minimize the number of physical network connections to a shared server. This is achieved by sharing a dispatcher's set of connections among multiple client processes.

Microsoft Transaction Server Connection Pooling

Microsoft Transaction Server provides a resource pooling infrastructure that enables certain resources to be pooled, such as memory and database connections.

OCI Connection Pooling

OCI connection pooling layer works with MTS resource pooling to provide pooled Oracle client/server sessions. The OCI connection pooling layer also caches Oracle Net connections to reduce client/server session setup time.

Working with In-Doubt Transactions

Oracle uses distributed transactions in the following configurations:

- Distributed database configurations, such as distributed updates using database links
- External transaction managers, such as Tuxedo and MS DTC, for coordinating transaction outcome

The two-phase commit protocol completes these transactions. During phase one, the transaction manager (TM) requests the various resource managers involved in the TM's transaction to prepare the underlying distributed transactions. In phase two, the TM determines whether it commits or terminates the transaction, and requests the resource managers to commit or terminate the underlying transaction. If a resource manager fails to receive the phase two notification, the underlying distributed transaction becomes in-doubt.

To integrate Oracle with Microsoft Transaction Server, distributed transactions are used in the database. Distributed transactions correspond to transactions coordinated by the MS DTC. A distributed transaction can become in-doubt when the transaction cannot commit or terminate (phase two of the two-phase commit). This occurs when the Microsoft Transaction Server application server process, database, or network fails.

> **See Also:**
>
> Managing Recovery Scenarios

Dropping the Microsoft Transaction Server Administrative User Account

The Microsoft Transaction Server administrative user account is created by running the `oramtsadmin.sql` script. If you later change the database with which Microsoft Transaction Server is coordinating transactions, you can drop the administrative user account schema from the previous database.

To drop the Microsoft Transaction Server administrative user account:

1. Start SQL*Plus:

   ```
   c:\> sqlplus /NOLOG
   ```

2. Connect to the database as `SYSDBA`:

   ```
   SQL> CONNECT / AS SYSDBA
   ```

3. Enter the following command to drop administrative user account schema:

   ```
   SQL> DROP USER mtsadmin_username CASCADE;
   ```

   where *`mtsadmin_username`* is the Microsoft Transaction Server administrative user account (default is `mtssys`).

> **See Also:**
>
> See Managing Recovery Scenarios for information on creating the Microsoft Transaction Server administrative user account for the new database

Glossary

Atomicity, Consistency, Isolation, and Durability (ACID)

ACID consists of the four primary attributes provided to any transaction by a transaction manager (also called a transaction manager).

component object model (COM)

A binary standard that enables objects to interact with other objects, regardless of the programming language in which each object was written.

distributed component object model (DCOM)

An extension of COM that enables objects to interact with other objects across a network.

data manipulation language

The category of SQL statements that query and update database data. Common DML statements are `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.

JOB_QUEUE_PROCESSES

This initialization parameter specifies the maximum number of `DBMS_JOB` jobs and Oracle Scheduler (`DBMS_SCHEDULER`) jobs running concurrently on an instance. This parameter must be set to at least 1 to run Oracle Scheduler or `DBMS_JOB` jobs and to use database features that depend on these jobs

listener.ora

A listener configuration file that identifies the following for a listener:

- Unique name
- Protocol addresses on which it accepts connection requests
- Services for which it is listening

Microsoft .NET

Microsoft .NET is a set of Microsoft software technologies used to connect information, people, systems, and devices through web services to each other and to larger applications over the Internet.

Microsoft application demo

An Oracle Call Interface (OCI) implementation of the Visual C++ Sample Bank package that ships with Microsoft Transaction Server on Windows.

Microsoft Distributed Transaction Coordinator (MS DTC)

The focal point of the transaction process is a component of Microsoft Transaction Server called Microsoft Distributed Transaction Coordinator (MS DTC).

Microsoft Transaction Server

A COM-based transaction processing system that runs on an Internet or network server.

mtssys

The default Microsoft Transaction Server administrator username. In releases prior to Oracle9*i* Database release 1 (9.0.1), this was the username for the OraMTS.

net service name

The name used by clients to identify an Oracle Net server and the specific system identifier (SID) or database for the Oracle Net connection. A net service name is mapped to a port number and protocol. A net service name is also known as a connect string, database alias, host string, or service name.

This also identifies the specific SID or database to which the connection is attaching, and not just the Oracle Net server.

OraMTS

Abbreviation for "Oracle Services for Microsoft Transaction Server".

Oracle Call Interface (OCI)

An application programming interface that enables you to manipulate data and schemas in a database. You compile and link an OCI program in the same way that you compile and link a nondatabase application. There is no requirement for a separate preprocessing or precompilation step.

Oracle Data Provider for .NET (ODP.NET)

Oracle Data Provider for .NET (ODP.NET) features optimized data access to the Oracle Database from a .NET environment. ODP.NET includes support for connection pooling, PL/SQL, LOBs, RefCursors, globalization/localization, proxy user authentication/ parameter array binding, named parameters, and safe type mapping between Oracle types and .NET types.

Oracle Fail Safe

Ensures that if a failure occurs on one cluster node, then the databases and applications running on that node fail over (move) automatically and quickly to a surviving node.

Oracle MTS Recovery Service

The Oracle MTS Recovery Service resolves in-doubt transactions on the computer that started the failed transaction. A scheduled recovery job for each Microsoft Transaction Server-enabled database lets the Oracle MTS Recovery Service resolve in-doubt transactions.

Oracle Open Database Connectivity (ODBC) Driver

Oracle ODBC Driver provides a standard interface that allows one application to access many different data sources. The application's source code does not have to be recompiled for each data source. A database driver links the application to a specific data source. A database driver is a dynamic link library that an application can invoke on demand to gain access to a particular data source. Therefore, the application can access any data source for which a database driver exists.

Oracle Provider for OLE DB

Interfaces that offer high performance and efficient access to Oracle data by applications, compilers, and other database components.

Oracle Services for Microsoft Transaction Server (OraMTS)

A component that provides full integration of the Oracle Database with Microsoft Transaction Server. This component enables you to develop and deploy COM-based applications using Microsoft Transaction Server.

Optimal Flexible Architecture (OFA)

A set of file naming and placement guidelines for Oracle software and databases.

resource manager (RM)

Microsoft Transaction Server enlists the database to act as a resource manager (RM) in the transaction process.

SYSDBA

A special database administration role that contains all system privileges with the `ADMIN OPTION` and the `SYSOPER` system privilege. `SYSDBA` also permits `CREATE DATABASE` actions and time-based recovery.

SYSOPER

A special database administration role that permits a database administrator to perform `STARTUP`, `SHUTDOWN`, `ALTER DATABASE OPEN/MOUNT`, `ALTER DATABASE BACKUP`, `ARCHIVE LOG`, and `RECOVER`, and includes the `RESTRICTED SESSION` privilege.

tnsnames.ora

A file that contains connect descriptors mapped to net service names. The file can be maintained centrally or locally for use by all or individual clients.

transaction identifiers (XIDs)

Identifies the client computer from which a transaction originated.

Index

J

L

M

N

O

P

R

S

T

U

V

W