# Oracle® COM Automation Feature

Developer's Guide

11*g* Release 1 (11.1)

**B31223-02**

September 2007

ORACLE®

Oracle COM Automation Feature Developer's Guide, 11*g* Release 1 (11.1)

B31223-02



Primary Author: Tulika Das

Contributors: Neeraj Gupta, Janis Greenberg, Eric Belden, Steven Caminez, Jagadish Changavi, Barmak Meftah, Valarie Moore, Neeraj Gupta, Vikhram Shetty, Sujith Somanathan, Alex Keh, Christian Shay, Riaz Ahmed

# Contents

# Preface

This document is your primary source of introductory, installation, post-installation configuration, and usage information for Oracle COM Automation Feature.

This document describes the features of Oracle Database for Windows that apply to the Windows 2000, Windows XP, and Windows Server 2003 operating systems.

This Preface contains these topics:



## Audience

*Oracle COM Automation Feature Developer's Guide* is intended for developers who develop solutions that use COM.

To use this document, you need familiarity with:

- Component Object Model (COM)
- OLE Automation
- Structured query language (SQL)
- Data definition language (DDL)
- Data manipulation language (DML)
- PL/SQL or Java
- Oracle object-relational database management system (ORDBMS) concepts
- Windows Server operating systems

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be

accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

### Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

### Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

### TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

## Related Documents

For more information, see these Oracle resources:

- *Oracle Database Installation Guide for Microsoft Windows*
- *Oracle Database Release Notes for Microsoft Windows*
- *Oracle Database Platform Guide for Microsoft Windows*
- *Oracle Services for Microsoft Transaction Server Developer's Guide*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database New Features Guide*
- *Oracle Database Concepts*
- *Oracle Database Reference*
- *Oracle Database Java Developer's Guide*

For information about Oracle error messages, see *Oracle Database Error Messages*. Oracle error message documentation is available only in HTML. If you only have access to the Oracle Documentation CD, you can browse the error messages by range. Once you find the specific range, use your browser's "find in page" feature to locate 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 about how these schemas were created and how you can use them yourself.

## 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. |

# 1

# Introducing Oracle COM Automation Feature

This chapter describes the Oracle COM Automation Feature Software Development Kit (SDK) and provides an overview of the product. Read this chapter before installing or using Oracle COM Automation Feature.

This chapter contains these topics:



## Overview of Oracle COM Automation Feature

Oracle COM Automation Feature enables you to use **Component Object Model (COM)**-based components to customize and enhance the functionality of the Oracle database on Windows operating systems.

You can build your own custom components or use the thousands of prebuilt components that are available from third-party independent software vendors (ISVs).

### Oracle COM Automation Feature Functionality

Oracle COM Automation Feature provides a mechanism to manipulate COM objects through either **PL/SQL** or Java.

- Oracle COM Automation Feature acts as a generic wrapper around the `IDispatch` interface.
- Oracle COM Automation Feature externalizes all methods supported by the `IDispatch` interface.
- **COM** objects expose properties, data attributes, and methods (functions that perform an action) to the developer.
- The `IDispatch` interface supports three basic operations for any COM object:
  - Gets the value of an exposed property
  - Sets the value of an exposed property
  - Invokes a method on an object

When an Oracle COM Automation Feature application programming interface (API) is invoked from PL/SQL or Java stored procedures, Oracle COM Automation Feature converts the parameters to the appropriate COM Automation data types and then invokes the corresponding `IDispatch` API with the converted parameters.

**See Also:** Chapter 3, "Oracle COM Automation Feature Core Functionality" for descriptions of the data types and APIs

### Oracle COM Automation Feature for PL/SQL

Oracle COM Automation Feature for PL/SQL provides a PL/SQL package and exposes a set of application programming interfaces (APIs) to instantiate COM objects. Developers can call these APIs from PL/SQL subprograms, stored procedures, stored functions, or triggers to manipulate COM objects.

There are no restrictions concerning where these COM objects reside. They can be local to the database or accessed remotely through the **Distributed Component Object Model (DCOM)**.

### Oracle COM Automation Feature for Java

Oracle COM Automation Feature for Java provides a set of Java APIs to instantiate COM objects. Developers can call these APIs from Java stored procedures, Java functions, or Java triggers to manipulate COM objects.

Oracle COM Automation Feature for Java does not allow in-process COM Automation servers. Developers can use `dllhost` to support in-process servers.

## Benefits of Oracle COM Automation Feature

Oracle COM Automation Feature is a powerful and enabling infrastructure technology for Oracle developers on Windows. It has the following advantages:

- Ease of Development

  Oracle COM Automation Feature exposes a simple set of APIs to manipulate COM objects. If you are familiar with COM and Microsoft Visual Basic, you can easily incorporate these APIs into your PL/SQL subprograms or Java programs.

- Reusability

  Oracle COM Automation Feature enables you to leverage prebuilt COM components that have been developed in-house or by third-party independent software vendors (ISVs). In addition, there are thousands of existing COM components from which you can choose. The COM component market is expanding rapidly and already offers solutions to many common programming problems.

- Flexibility and Extensibility

  You can use Oracle COM Automation Feature to customize and enhance the functionality of the database server. Through the use of COM components, the Oracle database can be customized to:

  – Exchange data among productivity applications, such as Microsoft Word, Microsoft Excel, and Microsoft PowerPoint.

  – Generate reports using Seagate Crystal Reports.

  – Send and receive e-mail with MAPI-compliant applications.

  The possibilities for customization and extensibility of the database server are limitless.

- Enhanced Integration

  Oracle COM Automation Feature enables you to deploy Oracle Database in a combined Oracle and Windows environment. You can be assured that Oracle

COM Automation Feature integrates fully with and capitalizes on the services that are exposed by Windows, Microsoft BackOffice applications, and Microsoft Office applications.

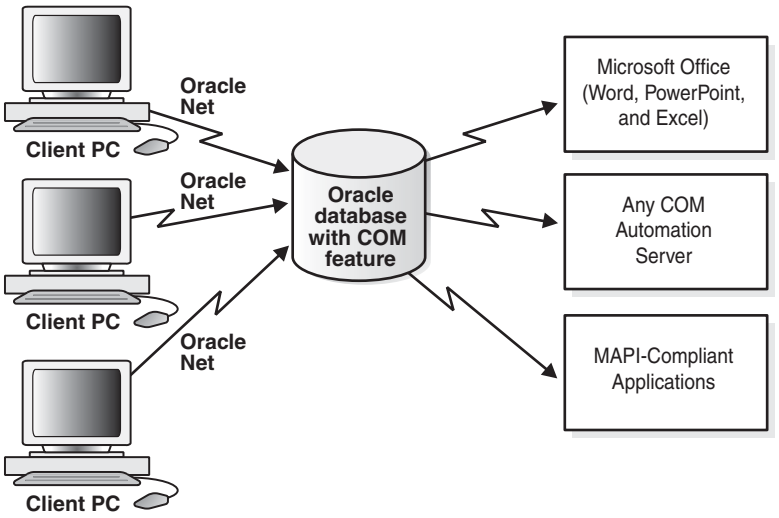
- Portability and Platform-Specific Requirements

  Applications using Oracle COM Automation Feature are written in Java or PL/SQL, which are platform-independent. Only the database instance that needs to invoke COM components must be run on Windows.

# Oracle COM Automation Feature Architecture

Figure 1–1 illustrates the interaction between an Oracle9*i* database with Oracle COM Automation Feature, client applications, and server applications.

***Figure 1–1 Oracle COM Interaction***




The architectural differences between Oracle COM Automation Feature for PL/SQL and for Java are described in the next two sections.

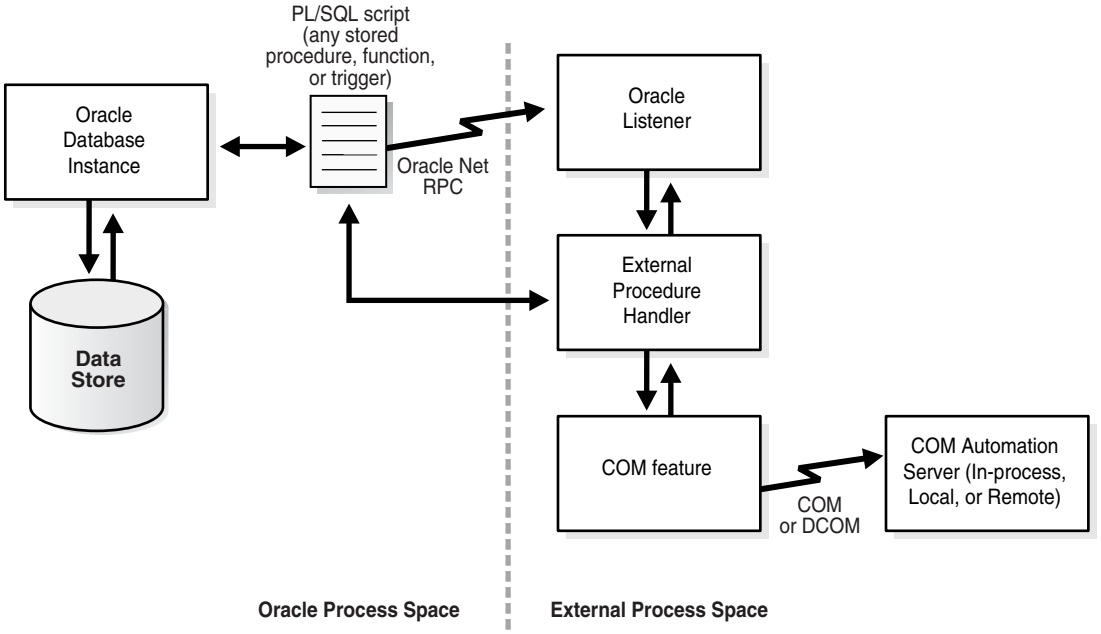
## PL/SQL Architecture

Oracle COM Automation Feature for PL/SQL provides a package of PL/SQL APIs for manipulating COM objects. These APIs are implemented as **external procedure**s in a **dynamic-link library (DLL)**.

Oracle9*i* supports external procedures that enable developers to call third-generation language (3GL) functions from server-based object type methods and stored procedures. External procedures are invoked exactly like standard PL/SQL stored procedures. However, unlike standard PL/SQL procedures where the body of the procedure is written in PL/SQL and stored in the database, external procedures are functions in the C programming language that reside within a DLL. You can invoke Oracle COM Automation Feature APIs in the same manner in which you call a standard PL/SQL stored procedure or function.

Figure 1–2 shows an Oracle9*i* database invoking COM Automation external procedure APIs.

***Figure 1–2 COM Automation Feature Architecture for PL/SQL***




## Invoking COM Automation External Procedure APIs

The database server invokes any of the COM Automation **external procedure** APIs as follows:

1. The PL/SQL interpreter looks up the path name to the Oracle COM Automation Feature DLL (`orawpcom`*`VER`*`.dll`) where *`VER`* is the release version.
2. The PL/SQL interpreter sends a message to the **listener** using **Oracle Net** to start `extproc.exe`, if it has not already been started for the current user session.
3. The PL/SQL interpreter passes the procedure name, the parameters, and the path name of the DLL to `extproc.exe`.
4. The `extproc.exe` file loads the DLL and executes the external procedure. Each of the COM Automation external procedure APIs in turn calls Win32 APIs that instantiate a COM object, set or get properties of a COM object, or invoke a method of a COM object.
5. `The extproc.exe` file acts as an intermediary and handles any interaction between Oracle COM Automation Feature and the database server.

## Architectural Impact on Availability Issues

The dependence on external procedures by Oracle COM Automation Feature for PL/SQL has implications for the availability of the database server.

You do not jeopardize the availability of the database server by using Oracle COM Automation Feature and custom or third-party COM objects in a production environment. Oracle COM Automation Feature operates outside of the Oracle kernel's address space. This safeguards the Oracle database from COM objects that stop abruptly.

## Java Architecture

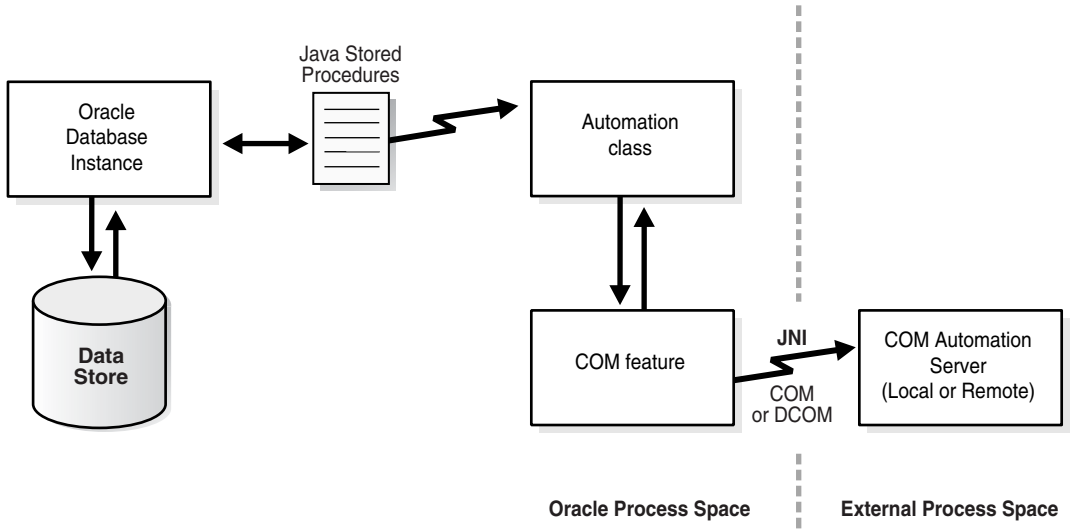
Oracle COM Automation Feature for Java is implemented by the Java Native Interface (JNI). The key components of this architecture are the `Automation` class and the Java COM Proxy DLL, `orawcom`*`VER`*`.dll`, where *`VER`* is the release version.

The interface is the `Automation` class, a Java proxy to the COM Automation server. The `Automation` class provides the methods necessary for developers to manipulate COM objects through the `IDispatch` interface.

The Java-specific COM proxy, `orawcom`*`VER`*`.dll`, enables Java functions to invoke their corresponding COM functions.

Figure 1–3 illustrates implementation of Oracle COM Automation Feature for Java.

***Figure 1–3 COM Automation Feature Architecture for Java***




### Reliability

Oracle COM Automation Feature for Java invokes COM components from the database server. However, these COM components are run outside of the Oracle9*i* database process. This design prevents unstable COM components from interfering with the database process.

# 2

# Installing and Configuring Oracle COM Automation Feature

This chapter provides an overview of the Oracle COM Automation Feature installation and postinstallation configuration tasks.

This chapter contains these topics:



## Oracle COM Automation Feature Components

The Oracle COM Automation Feature package is included as part of the Oracle installation. It contains the features and demos that illustrate how to use this product to solve real-world problems.

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

The COM Automation package includes the following **PL/SQL** and Java components:



### PL/SQL Components

The PL/SQL components for Oracle COM Automation Feature are:

- Oracle COM Automation Feature PL/SQL (`orawpcom`*`VER`*`.dll`)
- PL/SQL installation and definition script (`comwrap.sql`)
- Oracle COM Automation demonstration programs
- Message files (such as `comus.msb`)

Oracle COM Automation PL/SQL feature `orawpcom`*`VER`*`.dll` is located in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\bin` directory.

All other components are located in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\com` directory.

### Java Components

The Java components for Oracle COM Automation Feature are:

- The JAR file, `orawcom.jar`
- Oracle COM Automation Feature Java (`orawcom`*`VER`*`.dll`)
- Oracle COM Automation demonstration programs
- The `grant.sql` script file

Oracle COM Automation Java feature `orawcom`*`VER`*`.dll` is located in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\bin` directory. All other components are located in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\com\java` directory.

## System Requirements

Oracle COM Automation Feature requires:

- Windows XP, Windows 2000, or Windows Server 2003
- A functioning database on the computer before installation takes place

Note that you need to have a COM Automation server in the system to use Oracle COM Automation Feature. For example, the COM Automation Feature demos require that you first install the applications that are used in the demonstration programs:

- The Microsoft Word, Excel, and PowerPoint demos require Microsoft Office 2000 or later.
- The **messaging application programming interface (MAPI)** demo requires Microsoft Outlook 2000 or later.

The demonstrations and installations are discussed in "Overview of Oracle COM Automation Feature for PL/SQL Demos" on page 4-1 and "Overview of Oracle COM Automation Feature for Java Demos" on page 5-1.

## Upgrading from Oracle Database 10*g* to Oracle Database 11*g* Release 1

To upgrade Oracle COM Automation Feature from Oracle Database 10*g* Release 1 to Oracle Database 11*g* Release 1, do the following:

1. Rerun the `comwrap.sql` script.
2. Reinstall Java classes.
3. Run the `grant.sql` script.
4. Reinstall demos.

## Upgrading from Oracle9*i* to Oracle Database 10*g*

For this release, the right to execute `orawcom.dll` to `PUBLIC`, which was granted for Oracle9*i* must be revoked.

**Note:** In Oracle Database 10*g*, `orawcom.dll` is renamed to `orawcom`*VER*`.dll` where *VER* is the release version.

To upgrade Oracle COM Automation Feature from Oracle9*i* to Oracle Database 10*g*, do the following:

1. Rerun the `comwrap.sql` script.
2. Reinstall Java classes.
3. Revoke the right to execute `orawcom.dll` from `PUBLIC`.
4. Run the `grant.sql` script.
5. Reinstall demos.

**See Also:** "Configuring Oracle COM Automation Feature for PL/SQL" on page 2-3 for information about rerunning the `comwrap.sql` script

# Configurations for Oracle COM Automation Feature

Configuration procedures differ for PL/SQL and Java asexplained in the following sections:

- Configuring Oracle COM Automation Feature for PL/SQL
- Configuring Oracle COM Automation Feature for Java

## Configuring Oracle COM Automation Feature for PL/SQL

To configure Oracle COM Automation Feature for PL/SQL:

1. Start SQL*Plus.
2. Connect to the database as `SYSTEM`.

```
SQL> CONNECT SYSTEM@net_service_name
Enter password: password
```

3. Grant the `CREATE LIBRARY` privilege to the database users who will use Oracle COM Automation Feature. For example:

```
SQL> GRANT CREATE LIBRARY TO hr;
```

4. Connect to the user who will use Oracle COM Automation Feature, and run the `comwrap.sql` script at the SQL*Plus prompt:

```
SQL> CONNECT hr;
Enter password: password
SQL> @ORACLE_BASE\ORACLE_HOME\com\comwrap.sql
```

In the preceding command, *ORACLE_BASE*`\`*ORACLE_HOME* represents the Oracle home directory where Oracle COM Automation Feature is installed.

You will receive several `ORA-04043: object` *XXXX* `does not exist` messages when you run this script for the first time. These messages are usual.

## Configuring Oracle COM Automation Feature for Java

Perform the following to configure Oracle COM Automation Feature for Java:

1. Connect to the database as `SYSTEM` using SQL*Plus. For example:

```
SQL> CONNECT SYSTEM@net_service_name
Enter password: password
```

2. Run the `grant.sql` script with the name of the user who will be using Oracle COM Automation Feature. You may need to capitalize all letters in the user's name. For example:

```
SQL> @ORACLE_BASE\ORACLE_HOME\com\java\grant.sql HR
```

3. Run the `loadjava` tool at the command prompt as follows:

```
loadjava -force -resolve -user hr
     ORACLE_BASE\ORACLE_HOME\com\java\orawcom.jar
Password: password
```

   In the preceding command, `hr` is the user who uses Oracle COM Automation Feature.

> **See Also:** *Oracle Database Java Developer's Guide* for further information about the `loadjava` tool

# Configuring the Listener for PL/SQL

This section describes the specific configurations for the **listener.ora** and **tnsnames.ora** files when used with Oracle COM Automation Feature for PL/SQL.

> **Note:** Oracle COM Automation Feature for Java needs no special modifications to the **listener.ora** and **tnsnames.ora** files.

Because Oracle COM Automation Feature for PL/SQL relies on **listener** callouts, you must configure the listener and **Oracle Net** remote procedure call (RPC) mechanism for the feature to work.

The following are examples of **listener.ora** and **tnsnames.ora** files that can be used with interprocess communication (IPC) to invoke external stored procedures.

> **See Also:** Oracle Database Net Services Administrator's Guide for additional information about configuring the `listener.ora` and `tnsnames.ora` files for **external procedure**s

### listener.ora Configuration File

```
LISTENER =
(ADDRESS_LIST =
        (ADDRESS=
          (PROTOCOL= IPC)
          (KEY= EXTPROC0)
        )
)
STARTUP_WAIT_TIME_LISTENER = 0
CONNECT_TIMEOUT_LISTENER = 10
TRACE_LEVEL_LISTENER = off
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = ORCL)
    )
```

```
    (SID_DESC =
      (SID_NAME = plsextproc)
      (PROGRAM=extproc)
    )
  )
PASSWORDS_LISTENER = (oracle)
```

### tnsnames.ora Configuration File

```
EXTPROC_CONNECTION_DATA=
  (DESCRIPTION=
    (ADDRESS=
        (PROTOCOL=IPC)
(KEY=EXTPROC0)
    )
    (CONNECT_DATA=(SID=plsextproc))
    )
  )
```

## Troubleshooting Listener Problems

An "ORA-28575: unable to open RPC connection to **external procedure** agent" error message indicates one of two possible **listener** problems.

### Problem 1

**Problem:** The listener is not started.

**Action:** You must start the `Oracle`*`HOME_NAME`*`TNSListener` service from the Control Panel or the command prompt.

To start Oracle services from the Control Panel:

1. Choose **Start**, **Settings**, and then **Control Panel**.

   The Control Panel window appears.

2. Double-click **Services**.

   The **Services** dialog box appears.

3. Go to `Oracle`*`HOME_NAME`*`TNSListener` in the list and verify that it has a status of Started. If it does not, select it and click **Start**.

To start Oracle services from the command prompt:

Enter the following command:

```
C:\> net start service
```

In the preceding command, *service* is a specific service name, such as `Oracle`*`HOME_NAME`*`TNSListener`.

### Problem 2

**Problem:** The listener is not configured correctly.

**Action:** You must modify the `listener.ora` and `tnsnames.ora` files.

> **See Also:** "Configuring the Listener for PL/SQL" on page 2-4 for information about how to configure these files

# Support for DCOM

Oracle COM Automation Feature supports the use of **Distributed Component Object Model (DCOM)** to access remote **Component Object Model (COM)** objects over a network.

To authenticate the client's access to the remote computer, DCOM passes the appropriate security credentials to the remote computer. The remote computer validates the security credentials and allows DCOM to proceed.

These security credentials are based on the domain user's privileges associated with either the client's listener service or database service. Table 2–1 indicates the determining service for COM Automation for PL/SQL and Java.

***Table 2–1 Services That Determine Security Credentials***

| COM Automation Feature for... | Is Determined by This Service |
|---|---|
| PL/SQL | Listener |
| Java | Oracle database service |

# Configuring the Computer for DCOM

To use **DCOM**, you must configure security settings on the following:

- The computer that is running the database instance
- The computer that contains the remote **COM** object

## Configurations for the Computer Running the Database Instance

The configuration for the computer running the database instance requires setting the **listener** and the database service to the same domain user.

### Setting Services to a Domain User

In this procedure for setting a service to a domain user, the service to be set is selected in Step 3.

You must follow this procedure twice, once to set the **listener** and once to set the database service. The order is unimportant.

To set a service to a domain user:

1. Choose **Start**, **Settings,** and then **Control Panel**. The Control Panel window appears.
2. Double-click **Services**. The Services dialog box appears.
3. Select the service and click **Startup**. The service should be either `Oracle`*`HOME`*`_`*`NAME`*`TNSListener` or the database service.
4. Select the **This Account** option.
5. Enter the name or browse for a domain user.
6. Enter and confirm the password of the selected domain user.
7. Click **OK** to save the changes.

## Configuring the Computer Containing the Remote Object

Configuring the computer containing the remote object requires using the `dcomcnfg.exe` tool provided by Microsoft to configure the computer's DCOM security settings.

This tool enables you to set the access permissions, launch permissions, and configuration permissions for a specific COM object or all COM objects on a computer.

Using the `dcomcnfg.exe` tool, set the following:

1. Set the DCOM security privileges so that the appropriate service (that is, listener for PL/SQL and database service for Java), operating as a domain user, has sufficient privileges to instantiate and manipulate the remote COM object.
2. Set the remote COM object to execute with the same privileges as the service.

If the COM object attempts to perform an action for which it does not have permission, DCOM denies the operation and returns a security violation to Oracle COM Automation Feature. It is essential that you configure the DCOM security properly and provide the Oracle Database with the necessary permissions.

**See Also:** Microsoft documentation for more information aboutv:

- Using the `dcomcnfg.exe` tool and the implications of the related permissions
- Setting up the client and server computers to use DCOM

# 3

# Oracle COM Automation Feature Core Functionality

This chapter describes aspects of the programming interface for Oracle COM Automation Feature.

This chapter contains these topics:

- Data Type Conversions
- HRESULT Error Codes
- Oracle COM Automation for Java Exception Handling
- Typical COM Automation Functionality
- Application Programming Interfaces
- PL/SQL APIs
- Java APIs

## Data Type Conversions

Because Microsoft COM Automation uses COM Automation data types, and Oracle COM Automation Feature uses either PL/SQL or Java data types, Oracle COM Automation Feature must convert the data that it receives and pass it to the COM Automation object. Similarly, Oracle COM Automation Feature must pass the data that it receives from the COM Automation object and convert it.

### Data Type Conversion for PL/SQL

Table 3–1 shows the mapping between PL/SQL data types and COM Automation data types.

This guide follows a convention where COM Automation data types are prefaced by an initial *p* when used as `IN OUT` or `OUT` parameters. Data types without the initial *p* are `IN` parameters.

***Table 3–1 PL/SQL to COM Automation Data Types***

| PL/SQL Data Type | COM Automation Data Type |
|---|---|
| `VARCHAR2` | `BSTR, pBSTR` |
| `BOOLEAN` | `BOOL, pBOOL` |
| `BINARY_INTEGER` | `DISPATCH, pDISPATCH` |

*Table 3–1 (Cont.) PL/SQL to COM Automation Data Types*

| PL/SQL Data Type | COM Automation Data Type |
|---|---|
| DOUBLE PRECISION | UI1, pUI1, I2, pI2, I4, pI4, R4, pR4, R8, pR8, SCODE, pSCODE, CY, pCY, DISPATCH, pDISPATCH |
| DATE | DATE, pDATE |

**Note:** Oracle restricts a CY and pCY value to be between -9999999999.9999 and 9999999999.9999.

## Data Type Conversion for Java

Table 3–2 lists the supported COM Automation data types and related mappings to Java data types.

All data type mapping applies to properties, arguments, and return values, except void, which applies only to return values.

*Table 3–2 Java to COM Automation Data Types*

| Java Data Type | COM Automation Data Type |
|---|---|
| boolean | BOOL |
| char | CHAR |
| double | DOUBLE |
| int | INT |
| long | LONG |
| float | FLOAT |
| short | SHORT |
| byte | BYTE |
| java.lang.String | BSTR |
| oracle.win.com.Currency | CURRENCY |
| java.util.Calendar | DATE |
| void | VOID (return values only) |
| oracle.win.com.Automation | IDispatch* |

# HRESULT Error Codes

HRESULT error codes are provided by the Microsoft Windows API.

An HRESULT is a COM error code of the hexadecimal form 0x800*nnnnn*. However, it has the decimal form -214*nnnnnnn*. For example, passing an invalid object name when creating a COM object causes the HRESULT of -2147221005 to be returned, which is 0x800401f3 in hexadecimal form.

For complete information about the HRESULT return code, refer to the Microsoft documentation.

**See Also:** "Microsoft COM Automation Errors" on page A-3 for additional information

### PL/SQL Use of HRESULT

The PL/SQL APIs return an integer return code. The return code is `0` when successful, or a nonzero value of `HRESULT` when an error occurs.

> **See Also:** "GetLastError" on page 3-8 for additional information about how to interpret the return codes from Oracle COM Automation Feature

### Java Use of HRESULT

In the Java API, `HRESULT` is a data member of the `COMException` class.

> **See Also:** "Oracle COM Automation for Java Exception Handling" on page 3-3

## Oracle COM Automation for Java Exception Handling

Oracle COM Automation for Java uses standard Java exception mechanisms. Specifically, a Java exception class, `oracle.win.com.COMException`, is introduced to represent COM errors.

This exception is thrown by the `Automation` Java class when an error occurs.

The error information provided by this exception is similar to that provided by the PL/SQL API `GetLastError` function.

> **Note:** The `HRESULT` data member has the same meaning as the value of `HRESULT` returned by the PL/SQL functions.

If the COM error is `DISP_E_EXCEPTION` as indicated by the `excepInfo` data member, `COMException` uses the `source`, `description`, `helpfile`, and `helpid` data members. Otherwise, these data members are not valid.

The `COMException` writes an error message representing the COM error to the `errmsg` data member.

Table 3–3 lists the `COMException` data members and their descriptions.

***Table 3–3 COMException Data Members***

| Member | Description |
|---|---|
| `hresult` | is an `HRESULT` value as defined by the Windows API. |
| `errmsg` | is the textual representation of `HRESULT` in the appropriate language. |
| `source` | is the `source` of the exception, typically the application name. |
| `description` | is the error description. |
| `helpfile` | is the fully qualified path name of the `helpfile` containing more information about the error. |
| `helpid` | is the help context ID of a topic within the `helpfile` specified by help file. |
| `excepInfo` | is `DISP_E_EXCEPTION`, if HRESULT returns true, and `source`, `description`, `helpfile`, and `helpid` contain more information. |

### Code Sample

This example demonstrates the `COMException` exception.

```
try
  {
    // Some code that might throw a COMException exception.
  }
  catch(COMException e)
  {
    System.out.println(e.toString());
    if(e.excepInfo)
    {
      System.out.println(e.source);
      System.out.println(e.description);
      System.out.println(e.helpfile);
      System.out.println(e.helpid);
    }
  }
```

# Typical COM Automation Functionality

This section discusses the required information and the general steps to build a solution using Oracle COM Automation Feature.

## Information Required for COM Objects

Review the following information about the **COM** objects that you intend to use:

- You must determine the Program ID of the COM object. The Program ID, or **progID**, is a descriptive string that maps to the globally unique identifier (**GUID**), a hexadecimal number that uniquely identifies a COM object.

  The following string is an example of a progID:

  ```
  Excel.Worksheet.1
  ```

  Use the progID with the API that instantiates the COM object.

- You must be aware of the types of properties and methods that are exposed through the COM object's `IDispatch` interface. Usually, the ISV provides documentation describing the names and data type of the object's properties and the prototypes of the object's methods. Properties are referred to by a descriptive string, such as `xpos` or `ypos`. A property can be any standard COM Automation data type, such as `INT` or `BSTR`. The `GetProperty` and `SetProperty` APIs take the property name and a variable of the appropriate data type. Methods are referred to by a descriptive string, such as `InsertChart`. A method takes a set of parameters that are of different COM Automation data types and returns a COM Automation data type.

  The following is an example of a COM Automation method prototype in COM Interface Definition Language (IDL) grammar:

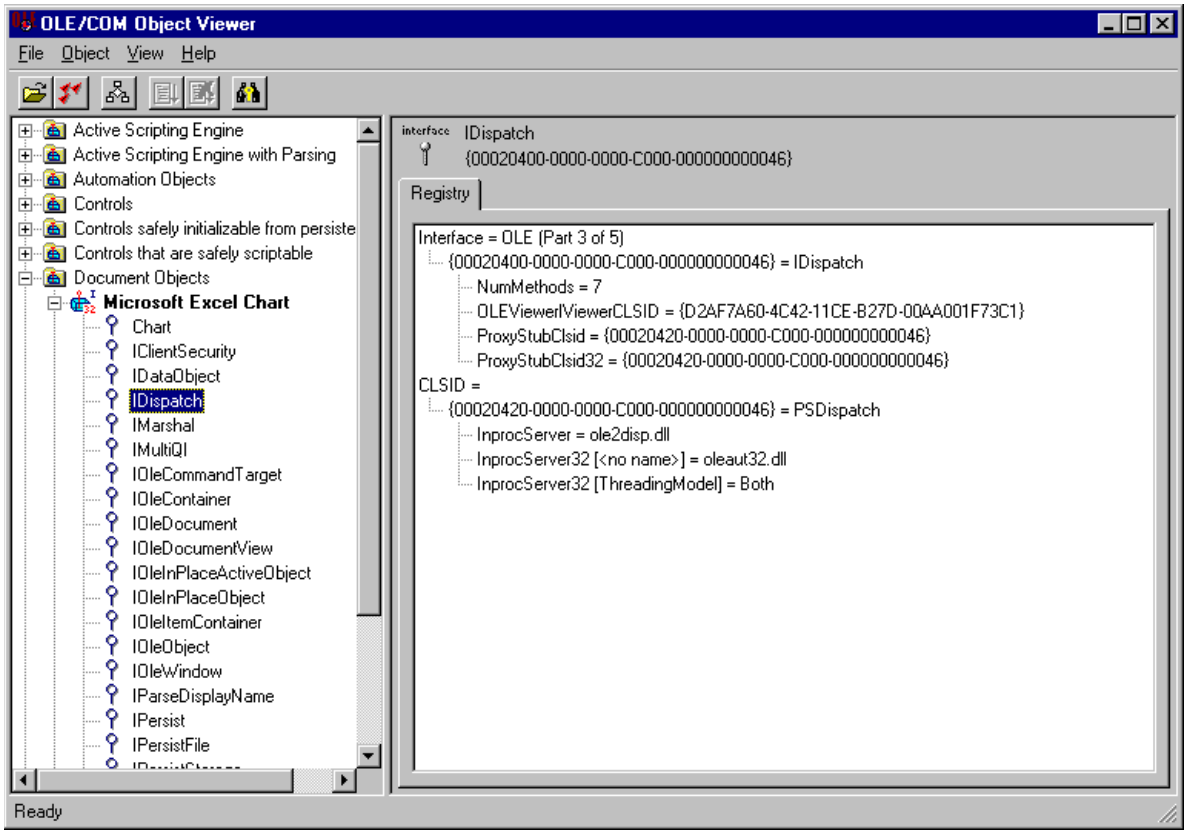
  ```
  [id(0x6003000)]
  long Post([in, out] long* lngAccountNo,
            [in, out] long* lngAmount,
            [in, out] BSTR* strResult);
  ```

  Interfaces define object methods and properties. COM IDL is used to specify interfaces that are defined on COM objects.

## OLE/COM Object Viewer

Microsoft provides a tool called the OLE/COM Object Viewer with Microsoft Visual Studio for browsing the properties and methods of COM objects on a local system. This tool enables you to quickly and easily determine the properties and methods that each COM object exposes. See Figure 3–1 for an example.

***Figure 3–1 OLE/COM Object Viewer***




## Using COM Automation Feature APIs

In a typical use of Oracle COM Automation Feature, you design a Java class or PL/SQL block to create and manipulate a COM object. The class or code block performs the following steps:

1. Creates the COM object as follows:
   - In PL/SQL, using `CreateObject`
   - In Java, using a constructor or the `Create` method
2. Manipulates the COM object calling the following APIs:
   - `GetProperty` to get a property value
   - `SetProperty` to set a property value to a new value
3. Calls `Invoke` to call a method

To prepare for the `Invoke` call, you use `InitArg` and `SetArg` to package the argument to be sent to the COM Automation method.

4. Calls `GetLastError` in PL/SQL to get the most recent error information
5. Destroys the object using `DestroyObject` in PL/SQL or `Destroy` in Java

# Application Programming Interfaces

This section lists and then describes the APIs available for Oracle COM Automation Feature.

## PL/SQL APIs

Oracle COM Automation Feature externalizes the following APIs for PL/SQL development:

- CreateObject
- DestroyObject
- GetLastError
- GetProperty
- SetProperty
- InitArg
- InitOutArg
- GetArg
- SetArg
- Invoke

## Java APIs

Oracle COM Automation Feature externalizes the following APIs for Java development:

- Automation Constructor
- Automation Methods
  - Create
  - Destroy
  - GetProperty
  - SetProperty
  - InitArg
  - SetArg
  - Invoke
- Currency Constructor
- Currency Methods
  - Get
  - Set

# PL/SQL APIs

This section describes the PL/SQL APIs for manipulating COM objects using the COM Automation interface. Each of the following PL/SQL stored procedures resides in the package ORDCOM.

## CreateObject

This API instantiates a COM object in a COM Automation server.

### Syntax

```
FUNCTION CreateObject(progid VARCHAR2, reserved BINARY_INTEGER, servername
VARCHAR2, objecttoken OUT BINARY_INTEGER) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| progid | the programmatic identifier (progID) of the COM Automation object to create. This character string describes the class of the COM Automation object and has the following form:<br>*COMComponent.Object*<br>*COMComponent* is the component name of the COM Automation server, and *Object* is the name of the COM Automation object. The specified COM Automation object must be creatable and must support the IDispatch interface. |
| reserved | a parameter currently reserved for future use. Pass a value of 0. Future versions of Oracle COM Automation Feature may use this parameter. |
| servername | the name of the remote DCOM server on which the COM object is being instantiated.<br>Passing a specified name forces Oracle COM Automation Feature to attempt to instantiate the COM object on a remote computer. Passing an empty string, for example, '', forces Oracle COM Automation Feature to check the registry for the location of the COM object. Registry information indicates whether the COM object is local or remote. Therefore, to create a local COM object, always pass an empty string and ensure that the registry indicates that the COM object exists locally. The registry information for COM objects can be configured with the tool dcomcnfg.exe. |
| objecttoken | the returned object token. It must be a local variable of data type BINARY_INTEGER. This object token identifies the created COM Automation object and is used in calls to the other Oracle COM Automation Feature APIs. |

### Remarks

The created COM Automation object is freed with a corresponding call to DestroyObject. This nullifies the internal representation of the object in the Oracle COM Automation Feature and releases all interfaces associated with the object.

This funtion returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### Code Sample

```
HRESULT BINARY_INTEGER;
applicationToken BINARY_INTEGER:=-1;
```

```
HRESULT :=ORDCOM.CreateObject('Excel.Application', 0, '', applicationToken);
IF (HRESULT!=0) THEN
  dbms_output.put_line(HRESULT);
END IF;
```

## DestroyObject

This API destroys a created COM Automation object.

### Syntax

```
FUNCTION DestroyObject(objecttoken BINARY_INTEGER) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `objecttoken` | the object token of a COM Automation object previously created by `CreateObject`. |

### Remarks

Calling `DestroyObject` nullifies the internal representation of the object in the Oracle COM Automation Feature and releases all interfaces associated with the object.

This function returns `0` when successful, or a nonzero value of `HRESULT` when an error occurs.

### Code Sample

```
HRESULT BINARY_INTEGER;
applicationToken BINARY_INTEGER:=-1;

/* Assume applicationToken is initialized. */

HRESULT:=ORDCOM.DestroyObject(applicationToken);
IF (HRESULT!=0) THEN
   dbms_output.put_line(HRESULT);
```

## GetLastError

This API obtains the COM Automation error information about the last error that occurred.

### Syntax

```
FUNCTION GetLastError(source OUT VARCHAR2, description OUT VARCHAR2, helpfile OUT
VARCHAR2, helpid OUT BINARY_INTEGER) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `source` | the source of the error information. If specified, it must be a local `CHAR` or `VARCHAR` variable. The return value is truncated to fit the local variable if necessary. |
| `description` | the description of the error. If specified, it must be a local `CHAR` or `VARCHAR` variable. The return value is truncated to fit the local variable if necessary. |

| Where | Is |
|---|---|
| helpfile | the Help file for the COM Automation object. If specified, it must be a local CHAR or VARCHAR variable. The return value is truncated to fit the local variable if necessary. |
| helpid | the Help file context ID. If specified, it must be a local INT variable. |

### Remarks

Each call to an Oracle COM Automation Feature API (except GetLastError) resets the error information, so that GetLastError obtains error information only for the most recent Oracle COM Automation Feature API call. Because GetLastError does not reset the last error information, it can be called multiple times to get the same error information.

This funtion returns 0 when successful, or a nonzero value of HRESULT when an error occurs.

See "Microsoft COM Automation Errors" on page A-3 for a description of the types of errors that can be returned by this function.

### Code Sample

```
HRESULT BINARY_INTEGER;
applicationToken BINARY_INTEGER:=-1;
error_src VARCHAR2(255);
error_description VARCHAR2(255);
error_helpfile VARCHAR2(255);
error_helpID BINARY_INTEGER;

HRESULT:=ORDCOM.CreateObject('Excel.Application', 0, '', applicationToken);
IF (HRESULT!=0) THEN
  ORDCOM.GetLastError(error_src, error_description, error_helpfile,
      error_helpID);
  dbms_output.put_line(error_src);
  dbms_output.put_line(error_description);

  dbms_output.put_line(error_helpfile);
END IF;
```

## GetProperty

This API returns the property value of a COM Automation object.

### Syntax

```
FUNCTION GetProperty(objecttoken BINARY_INTEGER, propertyname VARCHAR2, argcount
BINARY_INTEGER, propertyvalue OUT any_PL/SQL_data type) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| objecttoken | the object token of a COM object previously created by CreateObject. |
| propertyname | the property name of the COM object to return. |
| argcount | the index of the property array. If the property is not an array, then the developer should specify 0. |

| Where | Is |
|---|---|
| propertyvalue | the returned property value. The returned property type depends on the COM Automation data type that is returned. You must pass the PL/SQL data type that corresponds to the COM Automation data type of the COM Automation property. Otherwise, the COM Automation Feature will not properly convert the COM Automation data type. |
| *any_PL/SQL_data type* | any data type supported by COM Automation Feature. |

### Remarks

If the property returns a COM object, then you must specify a local variable of data type BINARY_INTEGER for the propertyvalue parameter. An object token is stored in the local variable, and this object token can be used with other COM Automation stored procedures.

When the property returns an array, if propertyvalue is specified, then it is set to NULL.

This function returns 0 when successful, or a nonzero value of HRESULT when an error occurs.

### Code Sample

```
/*
 * This is an excerpt from a Microsoft Excel application.
 */

HRESULT BINARY_INTEGER;
ChartObject BINARY_INTEGER := -1;
ChartToken BINARY_INTEGER := -1;

/* Assume ChartObject is initialized. */

HRESULT := ORDCOM.GetProperty(ChartObject, 'Chart', 0, ChartToken);
IF (HRESULT!=0) THEN
  dbms_output.put_line(HRESULT);
END IF;
```

## SetProperty

This API sets a property of a COM Automation object to a new value.

### Syntax

```
FUNCTION SetProperty(objecttoken BINARY_INTEGER, propertyname VARCHAR2, newvalue
any_PL/SQL_data type, data type VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| objecttoken | the object token of a COM Automation object previously created by CreateObject. |
| propertyname | the property name of the COM object to set to a new value. |
| newvalue | the new value of the property. It must be a value of the appropriate data type. |

| Where | Is |
|---|---|
| `data type` | the explicitly specified data type of the value passed in. The available data types are:<br>■ `UI1` - byte integer<br>■ `I2` - 2 byte integer<br>■ `I4` - 4 byte integer<br>■ `R4` - IEEE 4 byte real<br>■ `R8` - IEEE 8 byte real<br>■ `SCODE` - error code<br>■ `CY` - currency (value - 9999999999.9999 to 9999999999.9999) (This is an Oracle restriction)<br>■ `DISPATCH` - dispatch pointer<br>■ `BSTR` - String<br>■ `BOOL` - boolean<br>■ `DATE` - date |
| *`any_PL/SQL_data type`* | any data type supported by COM Automation Feature. |

### Remarks

This function returns a `0` when successful, or a nonzero value of `HRESULT` when an error occurs.

### Code Sample

```
/*
 * This is an excerpt from a Microsoft Excel application.
 */

HRESULT BINARY_INTEGER;
RangeToken BINARY_INTEGER := -1;

/* Assume RangeToken is initialized. */

HRESULT := ORDCOM.SetProperty(RangeToken, 'Value', 'EmpNo', 'BSTR');
IF (HRESULT!=0) THEN
  dbms_output.put_line(HRESULT);
END IF;
```

## InitArg

This API initializes the parameter set passed to an `Invoke` call.

### Syntax

```
PROCEDURE InitArg();
```

### Remarks

The `InitArg` call initializes the parameter set. After `InitArg` has been called, a `SetArg` call sets the first parameter to the specified value. A second `SetArg` call sets the second parameter in the parameter list. Subsequent calls set the *n*th parameters in the parameter list, where *n* is the number of times `SetArg` has been called after an

InitArg call. Another call to InitArg resets the argument list and a call to SetArg sets the first parameter again.

### Code Sample

See "Invoke" on page 3-15 for sample code.

## InitOutArg

InitOutArg must be called after a COM method is invoked in preparation for getting the values of OUT and IN OUT parameters using GetArg. After calling InitOutArg, the first call to GetArg gets the value for the first OUT or IN OUT parameter, the second call to GetArg gets the value for the second OUT or IN OUT parameters, and so on. Calling InitOutArg again restarts this process.

### Syntax

```
PROCEDURE InitOutArg();
```

### Remarks

See the section on SetArg data type strings in "SetArg" on page 3-21 for information about IN and OUT parameters.

### Code Sample

See "Invoke" on page 3-15 for sample code.

## GetArg

Gets the argument of OUT and IN OUT parameters after the COM method has been invoked.

### Syntax

```
PROCEDURE GetArg(data OUT any_PL/SQL_data type, type VARCHAR2);
```

| Where | Is |
|---|---|
| data | the value of the OUT or IN OUT parameter after the COM method has been invoked. |
| type | the COM Automation data type of the parameter. |

| Where | Is |
|---|---|
| | The available data types are:<br>■ pUI1 - byte integer<br>■ pI2 - 2 byte integer<br>■ pI4 - 4 byte integer<br>■ pR4 - IEEE 4 byte real<br>■ pR8 - IEEE 8 byte real<br>■ pSCODE - error code<br>■ pCY - currency (value -9999999999.9999 to 9999999999.9999) (This is an Oracle restriction)<br>■ pDISPATCH - dispatch pointer<br>■ pBSTR - String<br>■ pBOOL - Boolean<br>■ pDATE - date |
| *any_PL/SQL_data type* | any data type supported by COM Automation Feature. |

### Remarks

See the section on SetArg data type strings in "SetArg" on page 3-13 for information about IN and OUT parameters.

### Code Sample

See "Invoke" on page 3-15 for sample code.

## SetArg

Used to construct the parameter list for the next Invoke call.

SetArg sets a parameter's value to be passed by value.

### Syntax

```
PROCEDURE SetArg(paramvalue any_PL/SQL_data type, data type VARCHAR2);
```

| Where | Is |
|---|---|
| paramvalue | the value of the parameter to be passed to an Invoke call. The parameter set is the *n*th parameter in the parameter list, where *n* is the number of times SetArg has been called after an InitArg call. |
| data type | the explicitly specified data type for the parameters.<br>Those data types prefaced by an initial p are IN OUT or OUT parameters. The p indicates that the VT_BYREF flag will be set for the COM Automation data type. |

| Where | Is |
|---|---|
| | Those data types without the initial `p` are `IN` parameters. The available data types are:<br>■ `UI1` - byte integer<br>■ `pUI1` - byte integer<br>■ `I2` - 2-byte integer<br>■ `pI2` - 2-byte integer<br>■ `I4` - 4-byte integer<br>■ `pI4` - 4-byte integer<br>■ `R4` - IEEE 4-byte real<br>■ `pR4` - IEEE 4-byte real<br>■ `R8` - IEEE 8-byte real<br>■ `pR8` - IEEE 8-byte real<br>■ `SCODE` - error code<br>■ `pSCODE` - error code<br>■ `CY` - currency (value -9999999999.9999 to 9999999999.9999) (This is an Oracle restriction)<br>■ `pCY` - currency (value -9999999999.9999 to 9999999999.9999) (This is an Oracle restriction)<br>■ `DISPATCH` - dispatch pointer<br>■ `pDISPATCH` - dispatch pointer<br>■ `BSTR` - String<br>■ `pBSTR` - String<br>■ `BOOL` - Boolean<br>■ `pBOOL` - Boolean<br>■ `DATE` - date<br>■ `pDATE` - date |
| *`any_PL/SQL_data type`* | any data type supported by COM Automation Feature. |

## Remarks

Each `SetArg` procedure sets the *n*th parameter value. The `InitArg` call initializes the parameter set. After `InitArg` has been called, a `SetArg` call sets the first parameter to the specified value. A second `SetArg` call sets the second parameter in the parameter list. Subsequent calls set the *n*th parameters in the parameter list, where *n* is the number of times `SetArg` has been called after an `InitArg` call. Another call to `InitArg` resets the argument list and a call to `SetArg` sets the first parameter again.

Data types without the initial *p* are `IN` parameters. Those data types prefaced by an initial *p* are `IN OUT` or `OUT` parameters.

## Code Sample

See "Invoke" on page 3-15 for sample code.

## Invoke

This API calls a method of a COM Automation object. This function uses the parameter list, previously created by the calls to `InitArg` and `SetArg` as input for the COM Automation method.

### Syntax

```
FUNCTION Invoke(objecttoken BINARY_INTEGER, methodname VARCHAR2, argcount BINARY_
INTEGER, returnvalue OUT any_PL/SQL_data type) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `objecttoken` | the object token of a COM Automation object previously created by `CreateObject`. |
| `methodname` | the method name of the COM Automation object to call. |
| `argcount` | the number of arguments passed to the COM Automation object method. |
| `returnvalue` | the return value of the method of the COM Automation object. If specified, it must be a local variable of the appropriate data type. |
| *any_PL/SQL_data type* | any data type supported by COM Automation Feature. |

### Remarks

If the return value of the function is a COM object, then the developer must specify a local variable of data type `BINARY_INTEGER` for the `returnvalue` parameter. An object token is stored in the local variable, and this object token can be used with other Oracle COM Automation Feature APIs.

This function returns `0` when successful, or a nonzero value of `HRESULT` when an error occurs.

### Code Sample

```
/*
* Following is the IDL definition of the COM Automation method
* being called:
*
* HRESULT TestOutArg([in, out] short *x1,
* [in] short x2,
* [out] short *x3,
* [out, retval] short *x4);
*/

HRESULT BINARY_INTEGER := -1;
applicationToken BINARY_INTEGER := -1;
x1 DOUBLE PRECISION := 12;
x2 DOUBLE PRECISION := 7;
x3 DOUBLE PRECISION := 0;
x4 DOUBLE PRECISION := 0;

/* Assume applicationToken is initialized. */

ORDCOM.InitArg();
ORDCOM.SetArg(x1, 'pI2');
ORDCOM.SetArg(x2, 'I2');
ORDCOM.SetArg(x3, 'pI2');
```

```
HRESULT := ORDCOM.Invoke(applicationToken, 'TestOutArg', 3, x4);
IF (HRESULT!=0) THEN
  dbms_output.put_line(HRESULT);
END IF;

ORDCOM.InitOutArg();
ORDCOM.GetArg(x1, 'pI2');
ORDCOM.GetArg(x3, 'pI2');
```

# Java APIs

This section describes the Java APIs for manipulating COM objects using the COM Automation interface. These APIs are found in the `Automation` and `Currency` Java classes.

The `Automation` Java class provides access to COM objects that support COM Automation. With this Java class, you can create a COM object and obtain a pointer to the `IDispatch` interface for the COM object. You can then get and set properties on the COM object, as well as invoke methods (with or without arguments) on the COM object. This class provides a wrapper for the COM object, so there is no direct access to the COM object or to its `IDispatch` interface.

The `Currency` Java class represents the CURRENCY COM Automation data type. CURRENCY is a an 8-byte number where the last four digits represent the fractional part of the value. For example, the number 12345 actually represents the value 1.2345. CURRENCY has a range of (+/-)922337203685477.5807.

### COM Object Reference Counting

COM object interface reference counting is handled internally, and `IUnknown::AddRef()` and `IUnknown::Release()` are not exposed. The user cannot explicitly address COM object interfaces. The lifetime of a particular COM object starts when the associated Java constructor or Create method is invoked, and it is released when the associated Destroy method is invoked.

### Constructors and Destructors

Because the default constructor does not create a COM object, there are two approaches to creating a COM object:

- Instantiate the Java object using the default constructor, and call one of the Create methods. Which `Create` method you use depends on whether you want to specify the server name. Later, you must call the Destroy method to free the COM object.

  The `Create` method can be called at any time, but if a COM object was previously created through one of the nondefault constructors or the `Create` method, then you must first call the `Destroy` method.

- Instantiate the Java object using a nondefault constructor. Which nondefault constructor you use depends on whether you want to specify the server name. Later, you must call the Destroy method to free the COM object.

### Handling COM Object Errors

All COM errors are mapped to Java exceptions. Users can catch COM object errors through the Java exception handling mechanism.

**Note:** Oracle COM Automation Feature for Java does not allow in-process COM Automation servers. Developers can use `dllhost` to support in-process servers.

## Automation Constructor

This API creates a COM object.

### Syntax

```
public Automation()
public Automation(String progID)
public Automation(String progID, String serverName)
```

| Where | Is |
|---|---|
| `progID` | the programmatic identifier (progID) of the COM Automation object to create. This character string describes the class of the COM Automation object and has the following form:<br>*`COMComponent.Object`*<br>*`COMComponent`* is the component name of the COM Automation server, and *`Object`* is the name of the COM Automation object. The specified COM Automation object must be creatable and must support the `IDispatch` interface. |
| `serverName` | the name of the remote DCOM server on which the COM object is being instantiated.<br>Passing a specified name forces Oracle COM Automation Feature to attempt to instantiate the COM object on a remote computer. |

### Remarks

The default constructor `public Automation()` does nothing. It is used with a Create method.

Using a constructor that takes only the `progID` parameter forces Oracle COM Automation Feature to check the registry for the location of the COM object. Registry information indicates whether the COM object is local or remote.

COM Automation objects created using the nondefault constructors are freed with a corresponding call to Destroy. This nullifies the internal representation of the objects in Oracle COM Automation Feature and releases all interfaces associated with the objects.

Oracle COM Automation Feature for Java does not allow in-process COM Automation servers. Developers can use `dllhost` to support in-process servers.

The `COMException` exception is thrown if an error occurs.

### Code Sample

The following code sample demonstrates the nondefault constructors.

```
// Use the registry to determine where to create the COM object.
  Automation word = new Automation("Word.Basic");

  // Create the COM object on the specified server.
```

```
Automation excel = new Automation("Excel.Application",
                                  "\\ServerName");

// Free the COM objects.
word.Destroy();
excel.Destroy();
```

## Create

This API instantiates a COM object in a COM Automation server.

### Syntax

```
public void Create(String progID)
public void Create(String progID, String serverName)
```

| Where | Is |
|---|---|
| `progID` | the programmatic identifier (progID) of the COM Automation object to create. This character string describes the class of the COM Automation object and has the following form:<br>*`COMComponent.Object`*<br>*`COMComponent`* is the component name of the COM Automation server, and *`Object`* is the name of the COM Automation object. The specified COM Automation object must be creatable and must support the `IDispatch` interface. |
| `serverName` | the name of the remote DCOM server on which the COM object is being instantiated.<br>Passing a specified name forces Oracle COM Automation Feature to attempt to instantiate the COM object on a remote computer. |

### Remarks

The COM Automation object created with the `Create` method is freed with a corresponding call to `Destroy`. This nullifies the internal representation of the object in Oracle COM Automation Feature and releases all interfaces associated with the object.

Using the constructor that takes only the `progID` parameter forces Oracle COM Automation Feature to check the registry for the location of the COM object. Registry information indicates whether the COM object is local or remote.

Oracle COM Automation Feature for Java does not allow in-process COM Automation servers. Developers can use `dllhost` to support in-process servers.

The `COMException` exception is thrown if an error occurs.

### Code Sample

```
// Use the default constructor.
 Automation word = new Automation();
 Automation excel = new Automation();

 // Use the registry to determine where to create the COM object.
 word.Create("Word.Basic");

 // Create the COM object on the specified server system.
 excel.Create("Excel.Application", "\\ServerName");
```

```
// Free the COM objects.
word.Destroy();
excel.Destroy();
```

## Destroy

This API destroys a created COM Automation object.

### Syntax

```
public void Destroy()
```

### Remarks

Calling `Destroy` nullifies the internal representation of the object in the Oracle COM Automation Feature and releases all interfaces associated with the object.

### Code Sample

See "Create" on page 3-18 for code sample.

## GetProperty

This API gets a property value of a COM Automation object.

### Syntax

```
public allowed_type GetProperty(String propName, allowed_type[] propVal)
```

| Where | Is |
|---|---|
| `propName` | the property name of the COM object to return |
| `propVal` | the returned property value. The returned property type depends on the COM Automation type that is returned. The array must be big enough to hold at least one element although only the first element will be accessed to return the property. |
| *`allowed_type`* | from the following list:<br>■ `boolean`<br>■ `byte`<br>■ `char`<br>■ `short`<br>■ `int`<br>■ `long`<br>■ `float`<br>■ `double`<br>■ `java.long.String`<br>■ `oracle.win.com.Automation`<br>■ `oracle.win.com.Currency`<br>■ `java.util.Calendar` |

### Remarks

If the property is a COM object, then it can be retrieved using the *allowed_type* of `oracle.win.com.Automation`. The Automation Java object that is returned can be used to get and set properties and call methods on the property.

`GetProperty` uses an array parameter to return the property value to overload the `GetProperty` method. Overloading would not be possible if the property value were returned as a return value. The array solves the problem caused by Java not having an *out* parameter.

The property is still returned as a return value for convenience.

The `COMException exception` is thrown if an error occurs.

### Code Sample

```
// A Microsoft Excel ChartObject object.
Automation chartObject = null;
// A Microsoft Excel Chart object.
Automation chart = null;
// Used for properties of type Automation.
Automation[] autoProp = { null };

// Assume the Microsoft Excel ChartObject object is initialized.

// Get the Chart property.
chartObject.GetProperty("Chart", autoProp);
chart = autoProp[0];

// Set the Chart property.
chartObject.SetProperty("Chart", chart);
```

## SetProperty

This API sets a property of a COM Automation object to a new value.

### Syntax

`public void SetProperty(String propName, allowed_type propVal)`

| Where | Is |
|---|---|
| `propName` | the property name of the COM object being set to a new value |
| `propVal` | the new value of the property. It must be a value of the appropriate data type. |
| *allowed_type* | from the following list: |

| Where | Is |
|---|---|
| | ■ boolean<br>■ byte<br>■ char<br>■ short<br>■ int<br>■ long<br>■ float<br>■ double<br>■ java.long.String<br>■ oracle.win.com.Automation<br>■ oracle.win.com.Currency<br>■ java.util.Calendar |

### Remarks

If the property is a COM object, it can be set using the allowed type of `oracle.win.com.Automation`. The property value must be a valid Automation Java object.

The `COMException exception` is thrown if an error occurs.

### Code Sample

See "GetProperty" on page 3-19 for sample code.

## InitArg

This API initializes the parameter set passed to an `Invoke` call.

### Syntax

```
public void InitArg()
```

### Remarks

The `InitArg` call initializes the parameter set and must be called even if the COM method does not take any parameters. After `InitArg` has been called, a `SetArg` call sets the first parameter to the specified value. A second `SetArg` call sets the second parameter in the parameter list. Subsequent calls set the *n*th parameters in the parameter list, where *n* is the number of times `SetArg` has been called after an `InitArg` call. Another call to `InitArg` resets the argument list and a call to `SetArg` sets the first parameter again.

### Code Sample

See "Invoke" on page 3-22 for sample code.

## SetArg

This API is used to construct the parameter list for the next `Invoke` call.

### Syntax

```
public void SetArg(allowed_type val)
```

| Where | Is |
|---|---|
| `val` | the value of the parameter to be passed to an `Invoke` call. The parameter set is the *n*th parameter in the parameter list, where *n* is the number of times `SetArg` has been called after an `InitArg` call. |
| *`allowed_type`* | from the following list.<br>■ `boolean`<br>■ `byte`<br>■ `char`<br>■ `short`<br>■ `int`<br>■ `long`<br>■ `float`<br>■ `double`<br>■ `java.long.String`<br>■ `oracle.win.com.Automation`<br>■ `oracle.win.com.Currency`<br>■ `java.util.Calendar` |

### Remarks

If a parameter is a COM object, then the *`allowed_type`* of the corresponding argument should be `oracle.win.com.Automation`. The argument should be a valid Automation Java object.

No exceptions are thrown at this time. However, if an error occurs, for example, if the wrong argument type is passed, then it will be caught when the `Invoke` method is called.

### Code Sample

See "Invoke" on page 3-22 for sample code.

## Invoke

Calls a method of a COM Automation object. This function uses the parameter list, previously created by the calls to `InitArg` and `SetArg`, as input for the COM Automation method.

### Syntax

```
public void Invoke(String methodName, allowed_type[] retVal)
public void Invoke(String methodName)
```

| Where | Is |
|---|---|
| `methodName` | the method name of the COM Automation object to call |
| `retVal` | the return value of the method of the COM Automation object. If specified, then it must be a local variable of the appropriate data type. The array must be big enough to hold at least one element, although only the first element will be accessed to return the property. |

| Where | Is |
|---|---|
| *allowed_type* | a type from the following list:<br>■ boolean<br>■ byte<br>■ char<br>■ short<br>■ int<br>■ long<br>■ float<br>■ double<br>■ java.long.String<br>■ oracle.win.com.Automation<br>■ oracle.win.com.Currency<br>■ java.util.Calendar |

## Remarks

If the COM method returns a COM object as the return value, then the *allowed_type* of the return value is oracle.win.com.Automation. The Automation Java object that is returned can be used to get and set properties, and call methods on the return value.

To overload the Invoke method, Invoke uses an array parameter to return the values of COM object methods. Overloading would not be possible if the property value was returned as a return value. The array solves the problem caused by Java not having an *out* parameter.

The version of Invoke that takes only one parameter, public void Invoke(String methodName), is used for COM object methods with void return types.

The property is still returned as a return value for convenience.

The COMException exception is thrown if an error occurs.

## Code Sample

```
// A Microsoft Excel Worksheet object.
Automation workSheet = null;
// A Microsoft Excel ChartObjects collection object.
Automation chartObjects = null;
// A Microsoft Excel ChartObject object.
Automation chartObject = null;
// Used for return values of type Automation.
Automation[] autorv = { null };
// Dimensions for a Microsoft Excel ChartObject object.
short xpos = 100, ypos = 30, width = 400, height = 250;

// Assume the Microsoft Excel Worksheet object is initialized.

// Invoke a method that takes no arguments.
workSheet.InitArg();
workSheet.Invoke("ChartObjects", autorv);
chartObjects = autorv[0];

// Invoke a method that takes multiple arguments.
```

```
chartObjects.InitArg();
chartObjects.SetArg(xpos);
chartObjects.SetArg(ypos);
chartObjects.SetArg(width);
chartObjects.SetArg(height);
chartObjects.Invoke("Add", autorv);
chartObject = autorv[0];
```

## Currency Constructor

This API creates a `currency` Java object.

### Syntax

```
public Currency(long value)
```

| Where | Is |
|---|---|
| `value` | the 8-byte `CURRENCY` number |

## Get

This API gets the 8-byte `CURRENCY` number.

### Syntax

```
public long Get()
```

### Remarks

Returns the 8-byte `CURRENCY` number.

## Set

This API sets the 8-byte `CURRENCY` number.

### Syntax

```
public void Set(long value)
```

| Where | Is |
|---|---|
| `value` | the 8-byte `CURRENCY` number |

# 4

# Oracle COM Automation PL/SQL Demos

This chapter describes how to use Oracle COM Automation Feature demonstration programs for PL/SQL.

This chapter contains these topics:



## Overview of Oracle COM Automation Feature for PL/SQL Demos

Oracle COM Automation Feature for PL/SQL includes examples that demonstrate how to use the feature to build solutions. These demos provide base functionality and can serve as a foundation on which to build more customized, complex applications that use COM Automation. The demos are based on the human resources schema available with the sample schema.

Each demo exposes a core set of APIs that enables you to do simple operations using COM Automation. Each COM Automation server, such as Word and Excel, provides more advanced capabilities than what is offered through the demo APIs. To take advantage of these advanced features, you must design and code your own PL/SQL procedures.

In this release, COM Automation has provided the following demos:

- Microsoft Word Demo - Exchanges data between Microsoft Word and Oracle Database
- Microsoft Excel Demo - Exchanges data between Microsoft Excel and Oracle Database
- Microsoft PowerPoint Demo - Exchanges data between Microsoft PowerPoint and Oracle Database
- MAPI Demo - Exchanges data between Messaging Application Programming Interface (MAPI) compliant applications and Oracle Database

# Microsoft Word Demo

The following sections describe how to install the Microsoft Word demo and the APIs that it exposes. This demo is provided as an example of the types of solutions that can be built with Oracle Database and Microsoft Word.

The demo creates a Microsoft Word document containing the names of employees in the database.

The Microsoft Word demo provides the following:

- `ORDWord`, a PL/SQL package that exposes several APIs for manipulating Microsoft Word. This package is created by the `wordsol.sql` script.
- `worddem.sql`, a script that displays the capabilities of exchanging data between Oracle Database and Microsoft Word. It exchanges data from the `EMPLOYEES` and `JOBS` tables to a Microsoft Word document. These tables are available in the human resources schema in the sample schema.

## Installing the Microsoft Word Demo

Microsoft Word must be installed on the local computer before you install this demo.

To install Microsoft Word demos:

1. Start SQL*Plus.

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

2. Connect to the Oracle database instance as the user who will use the Microsoft Word demo. For example:

   ```
   SQL> connect hr
   Enter password: password
   ```

3. Run the `wordsol.sql` script at the SQL*Plus prompt:

   ```
   SQL> @ORACLE_BASE\ORACLE_HOME\com\demos\wordsol.sql;
   ```

   This script creates the `ORDWord` package in the current user's schema. You will receive the following error several times when you run this script for the first time:

   ```
   ORA-04043: object XXXX does not exist.
   ```

   These messages are normal.

## Using the Microsoft Word Demo

To use the Microsoft Word demo:

1. Run the `worddem.sql` script at the SQL*Plus prompt:

   ```
   SQL> @ORACLE_BASE\ORACLE_HOME\com\demos\worddem.sql;
   ```

   This script creates a Microsoft Word document (`worddemo.doc`) in the `C:\` directory. The document contains data from the `EMPLOYEES` and `JOBS` tables. These tables are available in the human resources schema in the sample schema.

2. Open the `worddemo.doc` file to see its contents.

## Core Functionality

The following subsections describe the APIs that the Microsoft Word demo exposes. These APIs are primitive and do not expose all the functionalities that Microsoft Word exposes through COM Automation.

### CreateWordObject

This API instantiates a `Word.Basic` object in the Microsoft Word Automation server.

#### Syntax

```
FUNCTION CreateWordObject() RETURN BINARY_INTEGER;
```

#### Remarks

This function must be called before any other operation can be performed. This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### FileNew

This API creates a new Microsoft Word document.

#### Syntax

```
FUNCTION FileNew() RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### FileLoad

This API loads a document into Microsoft Word.

#### Syntax

```
FUNCTION FileLoad(filename VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `filename` | the fully qualified filename of the document. |

#### Remarks

This function returns a `0` when successful or a nonzero `HRESULT` when an error occurs.

### FileSave

This API saves the current Microsoft Word document to disk.

#### Syntax

```
FUNCTION FileSave() RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### FileSaveAs

This API saves the current Microsoft Word document as a specific file.

#### Syntax

```
FUNCTION FileSaveAs(filename VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| filename | the fully qualified filename of the document. |

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### FileClose

This API closes the current Microsoft Word document.

#### Syntax

```
FUNCTION FileClose() RETURN BINARY_INTEGER;
```

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### InsertText

This API inserts a text string into the current Microsoft Word document.

#### Syntax

```
FUNCTION InsertText(textstr VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| textstr | the text that will be inserted into the document. |

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### InsertNewLine

This API inserts a new line into the current Microsoft Word document.

#### Syntax

```
FUNCTION InsertNewLine() RETURN BINARY_INTEGER;
```

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### FormatFontSize

This API sets the font size for the current Microsoft Word document.

#### Syntax

```
FUNCTION FormatFontSize(fontsize BINARY_INTEGER) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| fontsize | the point size of the font. |

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

## Microsoft Excel Demo

The following sections detail how to install the Microsoft Excel demo and describe the APIs that it exposes. This demo is provided as an example of the types of solutions that can be built with Oracle and Microsoft Excel.

The Microsoft Excel demo provides the following:

- ORDExcel, a PL/SQL package that exposes several APIs for manipulating Microsoft Excel. This package is created by the excelsol.sql script.
- exceldem.sql, a script that displays the capabilities of exchanging data between Oracle database instance and Microsoft Excel. It exchanges data from the EMPLOYEES and JOBS tables in Oracle database instance to a Microsoft Excel spreadsheet and puts it in a graph. These tables are available in the human resources schema in the sample schema.

### Installing the Microsoft Excel Demo

Microsoft Excel must be installed on the local computer before you install this demo.

To install the Microsoft Excel demo:

1. Start SQL*Plus.

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

2. Connect to the Oracle Database instance as the user who will use the Microsoft Excel demo. For example:

   ```
   SQL> connect hr
   Enter password: password
   ```

3. Run the excelsol.sql script at the SQL*Plus prompt:

   ```
   SQL> @ORACLE_BASE\ORACLE_HOME\com\demos\excelsol.sql;
   ```

   This script creates the ORDExcel package in the schema of the current user. You will receive the following error several times when you run this script for the first time:

   ```
   ORA-04043: object XXXX does not exist.
   ```

   These messages are normal.

## Using the Microsoft Excel Demo

To use the Microsoft Excel demo:

1. Run the `exceldem.sql` script at the SQL*Plus prompt:

   ```
   SQL> @ORACLE_BASE\ORACLE_HOME\com\demos\exceldem.sql;
   ```

   This script creates a Microsoft Excel spreadsheet (`excel`*`xxxxx`*`.xls`) in the `C:\` directory. The document contains data from the `EMPLOYEES` and `JOBS` tables. These tables are available in the human resources schema in the sample schema.

2. Open the `excel`*`xxxxx`*`.xls` file, where *xxxxx* is a time stamp, to see the content of this file.

## Core Functionality

The following subsections describe the APIs that the Microsoft Excel demo exposes. These APIs are primitive and do not expose all the functionalities that Microsoft Excel exposes through COM Automation.

### CreateExcelWorkSheet

This API starts the Microsoft Excel COM Automation server and instantiates the objects for a workbook and a worksheet.

#### Syntax

```
FUNCTION CreateExcelWorkSheet() RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### InsertData

This API inserts any kind of data into a specific cell of the current Excel worksheet.

#### Syntax

```
FUNCTION InsertData(range VARCHAR2, data any_PL/SQL_data type, data type VARCHAR2)
RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `range` | a string that indicates a specific cell in the current Excel worksheet (for example, 'A1', 'B1'). |
| `data` | the data that you want to insert into the current Excel worksheet. |
| `data type` | a string that indicates the data type of the data that you are inserting into Excel. The list of available data types are: |

| Where | Is |
|---|---|
| | ■ I2 - 2-byte integer<br>■ I4 - 4-byte integer<br>■ R4 - IEEE 4-byte real<br>■ R8 - IEEE 8-byte real<br>■ SCODE - error code<br>■ CY - currency<br>■ DISPATCH - dispatch pointer<br>■ BSTR - String<br>■ BOOL - boolean<br>■ DATE - date |
| *any_PL/SQL_data type* | any data type supported by COM Automation Feature. |

### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

## InsertChart

This API creates a chart of a specified range of data and inserts the chart at the x and y position of the current worksheet with the desired height and width.

### Syntax

```
FUNCTION InsertChart(xpos BINARY_INTEGER, ypos BINARY_INTEGER, width BINARY_
INTEGER, height BINARY_INTEGER, range VARCHAR2, type VARCHAR2) RETURN BINARY_
INTEGER;
```

| Where | Is |
|---|---|
| xpos | the x position in the current worksheet where the chart should be inserted |
| ypos | the y position in the current worksheet where the chart should be inserted |
| width | the width of the chart |
| height | the height of the chart |
| range | the range of cells to be graphed |
| type | the data type of the data to be graphed |

### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

## SaveExcelFile

This API saves the current Microsoft Excel workbook as a specific file.

### Syntax

```
FUNCTION SaveExcelFile(filename VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| filename | the fully qualified file name of the Excel workbook |

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### ExitExcel

Performs some cleanup and destroys the outstanding references to the Excel COM Automation server. This should be the last API called.

#### Syntax

```
FUNCTION ExitExcel() RETURN BINARY_INTEGER;
```

#### Remarks

This function returns a 0 when successful or a nonzero HRESULT when an error occurs.

## Microsoft PowerPoint Demo

The following sections detail how to install the Microsoft PowerPoint demo and describe the APIs that it exposes. This demo is provided as an example of the types of solutions that can be built with Oracle Database instance and Microsoft PowerPoint.

The Microsoft PowerPoint demo provides the following:

- ORDPPT, a PL/SQL package that exposes several APIs for manipulating Microsoft PowerPoint. This package is created by the pptsol.sql script.
- pptdem.sql, a script that displays the capabilities of exchanging data between Oracle Database instance and Microsoft PowerPoint. It exchanges data from the EMPLOYEES and JOBS tables in Oracle Database instance to a Microsoft PowerPoint document. These tables are available in the human resources schema in the sample schema.

### Installing the Microsoft PowerPoint Demo

Microsoft PowerPoint must be installed on the local computer before installing this demo.

To install the Microsoft PowerPoint demo:

1. Start SQL*Plus.

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

2. Connect to the Oracle Database instance as the user who will use the Microsoft PowerPoint demo. For example:

   ```
   SQL> connect hr
   Enter password: password
   ```

3. Run the pptsol.sql script at the SQL*Plus prompt:

   ```
   SQL> @ORACLE_BASE\ORACLE_HOME\com\demos\pptsol.sql;
   ```

This script creates the ORDPPT package in the current user's schema. You will receive the following error several times when you run this script for the first time:

```
ORA-04043: object XXXX does not exist.
```

These messages are normal.

## Using the Microsoft PowerPoint Demo

To run the Microsoft PowerPoint demo:

1. Run the pptdem.sql script at the SQL*Plus prompt:

   ```
   SQL> @ORACLE_BASE\ORACLE_HOME\com\demos\pptdem.sql;
   ```

   This script creates a Microsoft PowerPoint presentation (pptdemo.ppt) on C:\. The document contains a list of employee names.

2. Open pptdemo.ppt to see its contents.

## Core Functionality

The following subsections describe the APIs that the Microsoft PowerPoint demo exposes. These APIs are primitive and do not expose all the functionalities that Microsoft PowerPoint exposes through COM Automation.

### CreatePresentation

This API starts the Microsoft PowerPoint COM Automation server and instantiates the objects for a presentation.

#### Syntax

```
FUNCTION CreatePresentation (servername IN VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| servername | Microsoft Powerpoint COM Automation Server name |

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### AddSlide

This API inserts a new slide in the PowerPoint presentation.

#### Syntax

```
FUNCTION AddSlide (layout IN BINARY_INTEGER) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| layout | the layout of the new slide |

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### SetTitle

This API specifies the title of the PowerPoint slide.

#### Syntax

```
FUNCTION SetTitle (title IN VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `title` | Powerpoint slide title |

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### InsertText

This API inserts text into the specified location on the slide.

#### Syntax

```
FUNCTION InsertText (orientation IN BINARY_INTEGER, left IN BINARY_INTEGER, top IN
BINARY_INTEGER, width IN BINARY_INTEGER, height IN BINARY_INTEGER, text IN
VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `orientation` | orientation of the text box |
| `left` | distance between the left edge of the text box and the left edge of the slide in pixels |
| `top` | distance between the top edge of the text box and the top edge of the slide in pixels |
| `width` | width of the text box in pixels |
| `height` | height of the text box in pixels |
| `text` | text entered in the text box |

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### PresentationSave

This API saves the current PowerPoint presentation.

#### Syntax

```
FUNCTION PresentationSave RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### PresentationSaveAs

This API saves the current presentation using the specified name.

#### Syntax

```
FUNCTION PresentationSaveAs (filename IN VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `filename` | the fully qualified filename of the presentation. |

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### PresentationClose

This API closes the current PowerPoint presentation.

#### Syntax

```
FUNCTION PresentationClose RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### Exit

This API exits the PowerPoint program.

#### Syntax

```
FUNCTION Exit RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

## MAPI Demo

The following sections detail how to install the **messaging application programming interface (MAPI)** demo and describe the APIs that it exposes. This demo is provided as an example of the types of solutions that can be built with an Oracle Database instance and MAPI-compliant applications.

The MAPI demo provides the following:

- `ORDMAPI`, a PL/SQL package that exposes several APIs for manipulating the Extended MAPI client.
- `mapidem.sql`, a script that displays the capabilities of exchanging data between Oracle Database instance and the Extended MAPI client.

- `mapi.reg`, a registration entry file that updates the registry settings.

## Setting Up the Environment to Use the MAPI Demo

You must set up certain related applications to use the MAPI demo.

> **Note:** The following setup requires Microsoft Outlook 2000 or later. Outlook Express will not work.

To set up the environment for the MAPI demo:

1. Install Exchange Server and create a new account as follows:

   Select **Start**, **Programs**, **Microsoft Exchange,** and then A**ctive Directory Users and Computers**.

   Select your domain and expand the folders. Select users and right-click to create a new user.

2. Install Microsoft Outlook as follows:

   Select **Custom Install**. Select **Collaboration Data Objects**.

   > **Note:** During the installation, these are not installed by default.

   Select the **Corporate or Workgroup** option.

3. Configure Microsoft Outlook and set connection information as follows:

   Add the account that you created on Exchange Server.

   Enter your incoming and outgoing mail servers, and enter the account name and password.

   Select the connection type (for example, LAN).

4. Set Microsoft Outlook as the default program for the e-mail, newsgroups, and calendar tools as follows:

   From Internet Explorer, choose **Tools**, **Internet Options**, **Programs** and set the fields.

5. Patch `CDO.DLL` as mentioned in the MSDN article, 268272. This patched DLL is part of Microsoft Exchange 5.5. Manually copy the patched DLL to the proper location. The default location for this DLL is:

   ```
   C:\Program Files\Common Files\System\Mapi\1033\NT
   ```

6. Update the registry settings by double-clicking `MAPI.REG` from the Windows Explorer. `MAPI.REG` is located in:

   ```
   ORACLE_BASE\ORACLE_HOME\com\demos
   ```

## Preparing to Install MAPI Demo

The MAPI Solution invokes Extended MAPI client on behalf of the Oracle Database server. The Oracle Database service on Windows 2000 and higher, by default, runs as the system user `LocalSystem`. The MAPI profile for user `LocalSystem` is not easily

configured. Before using the MAPI Solution, change both the Oracle Database service and Oracle*HOME_NAME*TNSListenerservice to start up using a login user account.

To prepare to install the MAPI demo:

1. Log on to Windows using your local user account or domain user account, for example, DOMAIN-1\hr.
2. Start the MAPI server (for example, Microsoft Outlook) and configure the MAPI profile for the Windows 2000 and higher user DOMAIN-1\hr. Ensure that you can send out e-mail using this profile.
3. Go to Windows Control Panel/Services.
4. Shut down the Oracle*HOME_NAMETNS*Listener service.
5. Select the Oracle*HOME_NAMETNS*Listener service and click **Startup**.
6. Change the **Log On As** to **This Account** and fill in DOMAIN-1\hr.
7. Enter the password and confirm the password for DOMAIN-1\hr.
8. Restart the Oracle*HOME_NAMETNS*Listener service.
9. Shut down the Oracle Database service.
10. Select the Oracle Database service and click **Startup**.
11. Change **Log On As** to **This Account** and fill in DOMAIN-1\hr.
12. Enter the password and confirm the password for DOMAIN-1\hr.
13. Restart the Oracle Database service.

## Installing the MAPI Demo

The MAPI application, such as Microsoft Outlook 2000 or later, must be installed on the local computer before you install this demo.

To install the MAPI demo:

1. Start SQL*Plus.

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

2. Connect to the Oracle Database instance as the user who will use the MAPI demo. For example:

```
SQL> connect hr
Enter password: password
```

3. Run the mapisol.sql script at the SQL*Plus prompt:

```
SQL> @ORACLE_BASE\ORACLE_HOME\com\demos\mapisol.sql;
```

This script creates the ORDMAPI package in the current user's schema. You will receive the following error several times when you run this script for the first time:

```
ORA-04043: object XXXX does not exist.
```

These messages are normal.

## Using the MAPI Demo

To use the MAPI demo:

1. Open `mapidem.sql` with a text editor and change the e-mail address `hr@us.oracle.com` in `ORDMapi.AddRecipient` to your own e-mail address. If you are not using the default as your profile name, also change the profile name that is indicated in `ORDMapi.CreateMAPISession`, `MS Exchange Settings`. Save the changes.
2. Run the `mapidem.sql` script at the SQL*Plus prompt:

   ```
   SQL> @ORACLE_BASE\ORACLE_HOME\com\demos\mapidem.sql;
   ```

   This script connects to a database server, extracts the data, and sends an e-mail to a specified recipient.

## Core Functionality

The following subsections describe the APIs that the MAPI demo exposes. These APIs are primitive and do not expose all the functionalities that MAPI exposes through COM Automation.

### CreateMAPISession

This API starts the MAPI COM Automation server and instantiates the objects for a session.

#### Syntax

```
FUNCTION CreateMAPISession (servername IN VARCHAR2 DEFAULT '', profilename IN
VARCHAR2 DEFAULT NULL, password IN VARCHAR2 DEFAULT NULL) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| `servername` | MAPI server name |
| `profilename` | name of the profile present in the MAPI server |
| `password` | password to connect to the MAPI server |

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### NewMessage

This API creates a new message.

#### Syntax

```
FUNCTION NewMessage RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### AddRecipient

This API adds the e-mail address of a recipient. This is the address where the e-mail message will be sent.

#### Syntax

```
FUNCTION AddRecipient (emailaddress VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| emailaddress | e-mail address of the recipient |

#### Remarks

This function returns a 0 when successful or a nonzero HRESULT when an error occurs.

### SetSubject

This API specifies the subject of the e-mail message.

#### Syntax

```
FUNCTION SetSubject (subject VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| subject | the subject of the e-mail message |

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### SetBody

This API inserts the body text of the e-mail message.

#### Syntax

```
FUNCTION SetBody (messagetext VARCHAR2) RETURN BINARY_INTEGER;
```

| Where | Is |
|---|---|
| messagetext | the body of the e-mail message |

#### Remarks

This function returns 0 when successful, or a nonzero value for HRESULT when an error occurs.

### SendMessage

This API sends the e-mail message to the specified recipients.

#### Syntax

```
FUNCTION SendMessage RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

### EndMAPISession

This API exits the MAPI session.

#### Syntax

```
FUNCTION EndMAPISession RETURN BINARY_INTEGER;
```

#### Remarks

This function returns `0` when successful, or a nonzero value for `HRESULT` when an error occurs.

# 5

# Oracle COM Automation Java Demos

This chapter describes how to use the demonstration program designed for Oracle COM Automation Feature for Java.

This chapter contains these topics:



## Overview of Oracle COM Automation Feature for Java Demos

Oracle COM Automation Feature for Java includes an example that demonstrates how to use the feature to build solutions. The demo provides base functionality and can serve as a foundation on which to build more customized, complex applications that use COM Automation. This demo is based on the human resources schema available with the sample schema.

The demo exposes a core set of APIs that enable you to do simple operations using Oracle COM Automation Feature. Each COM Automation server, such as Word and Excel, provides more advanced capabilities than what is offered through the demo APIs. To take advantage of these advanced features, you must design and code your own Java classes.

In this release, COM Automation has provided the Microsoft Word Java Demo, which exchanges data between an Oracle Database instance and Microsoft Word.

## Microsoft Word Java Demo

The following sections describe how to install the Microsoft Word Java demo and the APIs that it exposes. This demo is provided as an example of the types of solutions that can be built with Oracle Database and Microsoft Word.

The demo creates a Microsoft Word document containing the names of employees in the database.

The Microsoft Word Java demo is installed in the *ORACLE_BASE**ORACLE_HOME*\com\java\demos directory and provides the following:

- `TestWORD.java`, the Java source for the demo. In addition to the collection of APIs, it includes the demo program `test`.
- `TestWORD.class`, the Java class for the demo.
- `TestWORD.sql`, the script that creates the call specification for the demo.

## Installing the Microsoft Word Java Demo

Microsoft Word must be installed on the local computer before you install this demo.

To install the demo:

1. Run the `loadjava` tool from the command line:

```
loadjava -force -resolve -user hr ORACLE_BASE\ORACLE_
HOME\com\java\demos\TestWORD.class
Password: password
```

2. Start SQL*Plus.

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

3. Connect to the Oracle Database instance as the user who will use the Microsoft Word demo. For example:

```
SQL> connect hr
Enter password: password
```

4. Run the `TestWORD.sql` script to create the call specification:

```
SQL> @ORACLE_BASE\ORACLE_HOME\com\java\demos\TestWORD.sql
```

> **See Also:** *Oracle Database Java Developer's Guide* for further information about the `loadjava` tool

## Using the Microsoft Word Java Demo

To use the Word demo:

1. Set `SERVEROUTPUT` on at the SQL*Plus prompt:

```
SQL> SET SERVEROUTPUT ON
```

2. Call `TestWORD()` at the SQL*Plus prompt:

```
SQL> CALL TestWORD();
```

   This creates a Microsoft Word document (`worddemoj.doc`) in the `C:\` directory. The document contains data from the `EMPLOYEES` and `JOBS` tables. These tables are available in the human resources schema in the sample schema.

3. Open `worddemoj.doc` to see its contents.

## Creating a Custom Application

The public class `TestWORD` API as described in "Core Functionality" on page 5-3, provides a wrapper around the `Word.Basic` COM Automation class as well as some sample code that demonstrates how to use the wrapper. This code was written to be run on the Oracle database server.

To create a custom application that uses this wrapper:

1. Instantiate an object of this class.
2. Create the `Word.Basic` object by calling the `CreateWordObject` method.
3. Create a new Microsoft Word document with the `FileNew` method, or open an existing document with the `FileLoad` method.
4. Use the `FormatFontSize`, `InsertText`, and `InsertNewLine` methods to add text and formatting to the document.

5. Save the document with the `FileSaveAs` or the `FileSave` method.
6. Call the `FileClose` method when you are finished with the document.
7. Call the `DestroyWordObject` method when you are finished with the `Word.Basic` object.

## Core Functionality

The following subsections describe the APIs that the Microsoft Word Java demo exposes. These APIs are primitive and do not expose all the functionalities that Microsoft Word exposes through COM Automation.

### TestWORD

This API is the constructor. It does nothing.

#### Syntax

```
public TestWORD()
```

### CreateWordObject

Creates the `Word.Basic` COM object.

#### Syntax

```
public void CreateWordObject(java.lang.String servername)
```

| Where | Is |
|---|---|
| servername | the server on which to create the COM object. Specify null or the empty string for the local server. |

### DestroyWordObject

This API destroys the `Word.Basic` COM object.

#### Syntax

```
public void DestroyWordObject()
```

### FileNew

This API creates a new Microsoft WORD document.

#### Syntax

```
public void FileNew()
```

#### Remarks

This API is a wrapper for the `FileNewDefault` COM method of the `Word.Basic` COM object.

### FileLoad

This API loads an existing Microsoft WORD document.

#### Syntax

```
public void FileLoad(java.lang.String filename)
```

| Where | Is |
|---|---|
| filename | the name of the file to load. |

#### Remarks

This API is a wrapper for the FileOpen COM method of the Word.Basic COM object.

### FormatFontSize

This API sets the font size.

#### Syntax

```
public void FormatFontSize(long fontsize)
```

| Where | Is |
|---|---|
| fontsize | the new font size. |

#### Remarks

This API is a wrapper for the FormatFont COM method of the Word.Basic COM object.

### InsertText

This API inserts text into the Microsoft Word document.

#### Syntax

```
public void InsertText(java.lang.String textstr)
```

| Where | Is |
|---|---|
| textstr | the text to insert. |

#### Remarks

This API is a wrapper for the Insert COM method of the Word.Basic COM object.

### InsertNewLine

This API inserts a new line into the Microsoft Word document.

#### Syntax

```
public void InsertNewLine()
```

#### Remarks

This API is a wrapper for the InsertPara COM method of the Word.Basic COM object.

### FileSaveAs

This API saves the Microsoft Word document using a specified name.

#### Syntax

```
public void FileSaveAs(java.lang.String filename)
```

| Where | Is |
|---|---|
| `filename` | the name of the file. |

#### Remarks

This API is a wrapper for the `FileSaveAs` COM method of the `Word.Basic` COM object.

### FileSave

This API saves the Microsoft Word document.

#### Syntax

```
public void FileSave()
```

#### Remarks

This API is a wrapper for the `FileSave` COM method of the `Word.Basic` COM object.

### FileClose

This API closes the Microsoft Word document, and exits Microsoft Word.

#### Syntax

```
public void FileClose()
```

#### Remarks

This API is a wrapper for the `FileClose` and `FileExit` COM methods of the `Word.Basic` COM object.

# A

# COM Automation Error Messages

This appendix contains these topics:



## Oracle COM Automation Feature, PL/SQL Errors

The following is a list of Oracle COM Automation Feature PL/SQL errors and their common causes.

**COM-0001: Not a Boolean type**

**Cause:** The property type, or return value type, is not a boolean, but a Boolean value was requested.

**Action:** Make sure that the variable is of the appropriate data type.

**COM-0002: Invalid Token or no interface for token**

**Cause:** The token that was specified does not reference any COM object created using `CreateObject`, or the COM object was freed using `DestroyObject`.

**Action:** Make sure that the interface exists.

**COM-0003: Maximum Objects reached**

**Cause:** Only 1024 COM objects can be active at any time. This includes COM objects created using `CreateObject` as well as COM objects obtained as property values and return values.

**Action:** Make sure that objects are destroyed after they are used, by calling DestroyObject.

**COM-0004: The registered CLSID for the ProgID is invalid**

**Cause:** The ProgID is located in the registry, but the CLSID associated with the ProgID is not correct.

**Action:** Check that the COM component of the specified ProgID is registered.

**COM-0005: An error occurred writing the CLSID to the registry**

**Cause:** The ProgID is not located in the registry. An attempt was made to create the ProgID and assign a CLSID to it, but the registry could not be modified.

**Action:** Ensure that your registry can be written to and is not corrupted.

**COM-0006: A specified class is not registered in the registration database**

**Cause:** A specified class is not registered in the registration database.

**Action:** Make sure that the class is registered.

**COM-0007: Failed to initialize COM Automation object**

**Cause:** There was an error creating the COM object.

**Action:** Make sure that the object is registered as a COM Automation object.

**COM-0008: No interface is supported**

**Cause:** This COM object does not support the `IDispatch` interface, so it cannot support COM Automation.

**Action:** Verify that the interface specified is valid.

**COM-0014: Failure to invoke**

**Cause:** There was an error invoking the method or property.

**Action:** Verify that the method name is valid for the object.

**COM-0015: Bad parameter count**

**Cause:** The number of parameters given for the method or property is different from the number of parameters expected.

**Action:** Make sure that the number of parameters for a method is equal to the count.

**COM-0017: The application needs to raise an exception. The structure passed in `pexcepinfo` should be filled in**

**Cause:** The COM object threw an exception.

**Action:** The exception includes an error source, error description, Help file, and the help file context. Call `GetLastError` to get this additional information

**COM-0018: The requested member does not exist, or the call to Invoke tried to set the value of a read-only property**

**Cause:** The requested member does not exist, or the call to `Invoke` tried to set the value of a read-only property.

**Action:** Make sure that the property value can be written to or the member exists.

**COM-0020: One of the arguments in rgvarg could not be coerced to the specified type**

**Cause:** One of the arguments is not the type expected by the method or property, and the argument cannot be coerced to the expected type.

**Action:** Make sure that the coerced arguments are of compatible data types.

**COM-0022: One or more of the arguments could not be coerced**

**Cause:** One of the arguments is not the type expected by the method or property, and the argument cannot be coerced to the expected type.

**Action:** Make sure that your arguments are compatible.

**COM-0025: Not an optional parameter**

**Cause:** A required argument is missing.

**Action:** Make sure that your argument count is correct for the number of parameters passed in.

**COM-0026: Name exceeded the maximum character allowed**

**Cause:** The property name, method name, server name, or ProgID is too long.

**Action:** Enter less than 1024 characters for the name.

# Microsoft COM Automation Errors

The following is a list of Microsoft COM Automation errors and their common causes. Both the hexadecimal and binary error codes are listed.

**(0x800401f3) (-2147221005)** ***Invalid class string***

**Cause:** The specified ProgID or CLSID is not registered as a COM object in the registry of the local computer.

**Action:** Correctly install the COM component.

**(0x8007007e) (-2147024770)** ***The specified module could not be found***

**Cause:** The specified COM object is registered as an in-process COM server (DLL file), but the DLL file could not be found or loaded.

**Action:** Correctly install the COM component.

**(0x80020004) (-2147352572)** ***Parameter not found***

**Cause:** A named parameter was specified before a positional parameter.

**Action:** Ensure that all named parameters are specified after all positional parameters.

**(0x80020005) (-2147352571)** ***Type mismatch***

**Cause:** The data type of a PL/SQL local variable used to store a returned property value or a method return value did not match the Visual Basic data type of the property or method return value, or the return value of a method was requested, but it does not return a value.

**Action:** Ensure that the local variable is of the appropriate data type and, for methods, ensure that the return value is not type void.

**(0x80020006) (-2147352570)** ***Unknown name***

**Cause:** The specified property or method name was not found for the specified object.

**Action:** Verify that the method or property name is valid for the object.

**(0x80020008) (-2147352568)** ***Bad variable type***

**Cause:** The data type of a PL/SQL or Java value passed as a method parameter did not match the COM Automation data type of the method parameter, or a `NULL` value was passed as a method parameter.

**Action:** Ensure that any local variables used as method parameters are of the appropriate data type and are set to a value other than `NULL`.

**(0x80080005) (-2146959355)** ***Server execution failed***

**Cause:** The specified COM object is registered as a local COM server (.EXE file), but the .EXE file could not be found or started.

**Action:** Correctly install the COM component.

# Glossary

**Component Object Model (COM)**

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

**Distributed Component Object Model (DCOM)**

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

**dynamic-link library (DLL)**

An executable file that a Windows application can load when needed

**external procedure**

A function written in a third-generation language (3GL), such as C, and callable from within PL/SQL or SQL as if it were a PL/SQL function or procedure

**GUID**

An identifier that uniquely identifies a COM object. GUID is an acronym for Globally Unique Identifier

**IID**

A GUID that identifies a COM interface

**listener**

The server process that listens for and accepts incoming connection requests from client applications. Oracle listener processes start up Oracle Database processes to handle subsequent communications with the client

**listener.ora**

A configuration file that describes one or more Transparent Network Substrate (TNS) listeners on a server

**messaging application programming interface (MAPI)**

A messaging architecture composed of a set of common application programming interfaces that enables multiple applications to interact with multiple messaging systems across a variety of hardware platforms

**Optimal Flexible Architecture (OFA)**

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

**Oracle COM Automation Feature**

An Oracle feature that enables PL/SQL developers to programmatically manipulate COM objects through the `IDispatch` COM Automation interface

**Oracle Net**

The Oracle client/server communication software that offers transparent operation to Oracle tools or databases over any type of network protocol and operating system

**PL/SQL**

Oracle's procedural language extension to SQL

**progID**

A descriptive string that maps to a GUID

**tnsnames.ora**

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

# Index

## G



## H



## I



## J



## L



## M



## O

## P



## R



## S



## T



## U



## W