Oracle® COM Automation Feature

Developer's Guide

Release 9.2

March 2002

Part No. A95499-01

ORACLE®

Oracle COM Automation Feature Developer's Guide, Release 9.2

Part No. A95499-01

Contributors: Janis Greenberg, Eric Belden, Steven Caminez, Jagadish Changavi, Barmak Meftah

Contents

Send Us Your Comments vii

Preface ix

Audience x
Organization x
Related Documentation xi
Conventions xii
Documentation Accessibility xviii

What's New in Oracle COM Automation Feature? xix

New Features in Oracle COM Automation Feature for Release 9.2 xx
New Features in Oracle COM Automation Feature for Release 9.0.1 xx
New Features in Oracle COM Automation Feature for Release 8.1.6 xxi

1 Introducing Oracle COM Automation Feature

Introduction to Oracle COM Automation Feature 1-2
Oracle COM Automation Feature Functionality 1-2
Oracle COM Automation Feature for PL/SQL 1-2
Oracle COM Automation Feature for Java 1-3
Benefits of Oracle COM Automation Feature 1-3
Oracle COM Automation Feature Architecture 1-5
PL/SQL Architecture 1-5
Invoking COM Automation External Procedure APIs 1-6
Architectural Impact on Availability Issues 1-7

Java Architecture 1-7
Reliability 1-8

2 Installing and Configuring Oracle COM Automation Feature

Oracle COM Automation Feature Components 2-2
System Requirements 2-3
Migration from Oracle8*i* to Oracle9*i* 2-3
Configurations for Oracle COM Automation Feature 2-4
Configuring Oracle COM Automation Feature for PL/SQL 2-4
Configuring Oracle COM Automation Feature for Java 2-4
Configuring the Listener for PL/SQL 2-5
Troubleshooting Listener Problems 2-6
Support for DCOM 2-7
Configurations for the Computer Running the Database Instance 2-8
Setting Services to a Domain User 2-8
Configuring the Computer Containing the Remote Object 2-8

3 Oracle COM Automation Feature Core Functionality

Datatype Conversions 3-2
Datatype Conversion for PL/SQL 3-2
Datatype Conversion for Java 3-2
HRESULT Error Codes 3-3
Oracle COM Automation for Java Exception Handling 3-4
Typical COM Automation Functionality 3-6
Information Required for COM Objects 3-6
OLE/COM Object Viewer 3-7
Using COM Automation Feature APIs 3-8
Application Programming Interfaces 3-9
PL/SQL APIs 3-10
CreateObject 3-10
DestroyObject 3-12
GetLastError 3-12
GetProperty 3-14
SetProperty 3-15
InitArg 3-17

InitOutArg 3-17
GetArg 3-18
SetArg 3-19
Invoke 3-21
Java APIs 3-22
Automation Constructor 3-24
Create 3-25
Destroy 3-26
GetProperty 3-27
SetProperty 3-28
InitArg 3-29
SetArg 3-30
Invoke 3-31
Currency Constructor 3-33
Get 3-34
Set 3-34

4 Oracle COM Automation PL/SQL Demos

Overview of PL/SQL Demos 4-2
Microsoft Word Demo 4-2
Installing the Microsoft Word Demo 4-3
Using the Microsoft Word Demo 4-3
Core Functionality 4-3
Microsoft Excel Demo 4-7
Installing the Microsoft Excel Demo 4-7
Using the Microsoft Excel Demo 4-8
Core Functionality 4-8
Microsoft PowerPoint Demo 4-11
Installing the Microsoft PowerPoint Demo 4-11
Using the Microsoft PowerPoint Demo 4-12
Core Functionality 4-12
MAPI Demo 4-15
Setting Up the Environment to Use the MAPI Demo 4-15
Preparing to Install MAPI Demo 4-16
Installing the MAPI Demo 4-17

Using the MAPI Demo 4-17
Core Functionality 4-18

5 Oracle COM Automation Java Demos

Oracle COM Automation Feature Java Demos Overview 5-2
Microsoft Word Java Demo 5-2
Installing the Microsoft Word Java Demo 5-3
Using the Microsoft Word Java Demo 5-4
Creating a Custom Application 5-4
Core Functionality 5-4

A COM Automation Error Messages

Oracle COM Automation Feature, PL/SQL Errors A-2
Microsoft COM Automation Errors A-4

Glossary

Index

Send Us Your Comments

Oracle COM Automation Feature Developer's Guide, Release 9.2

Part No. A95499-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: ntdoc_us@oracle.com
- FAX: (650) 506-7365 Attn: Oracle Database for Windows Documentation
- Postal service:
  Oracle Corporation
  Oracle Database for Windows Documentation Manager
  500 Oracle Parkway, Mailstop 1op6
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

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

This manual describes only the features of Oracle9*i* for Windows NT software that apply to the Windows NT, Windows 2000, Windows XP, and Windows 98 operating systems. Information on Oracle9*i* Personal Edition software on Windows 98 is not covered in this manual.

This preface contains these topics:

- Audience
- Organization
- Related Documentation
- Conventions
- Documentation Accessibility

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)
- Object Linking and Embedding (OLE) Automation
- Structured Query Language (SQL)
- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- PL/SQL or Java
- Oracle object-relational database management (ORDBMS) concepts
- Windows NT

Organization

This document contains:

Chapter 1, "Introducing Oracle COM Automation Feature"

Provides an overview of Oracle COM Automation Feature and Oracle Server architecture. Read this chapter *before* installing or using Oracle COM Automation Feature.

Chapter 2, "Installing and Configuring Oracle COM Automation Feature"

Describes how to install Oracle COM Automation Feature and the configuration tasks you *must* perform before using it. This chapter also lists the contents of the Oracle COM Automation Feature SDK and describes the system requirements.

Chapter 3, "Oracle COM Automation Feature Core Functionality"

Describes the core functionality of Oracle COM Automation Feature, including PL/SQL and Java APIs for manipulating COM objects.

Chapter 4, "Oracle COM Automation PL/SQL Demos"

Describes how to use the PL/SQL demos for Oracle COM Automation Feature.

Chapter 5, "Oracle COM Automation Java Demos"

Describes how to use the Java demos for Oracle COM Automation Feature.

Appendix A, "COM Automation Error Messages"

Describes the Oracle PL/SQL COM Automation Feature error codes and the Microsoft COM Automation error codes.

Glossary

Defines terms used in this document.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Database Installation Guide for Windows*
- *Oracle9i Database Release Notes for Windows*
- *Oracle9i Database Administrator's Guide for Windows*
- *Oracle Enterprise Manager Administrator's Guide*
- *Oracle Services for Microsoft Transaction Server Developer's Guide*
- *Oracle9i Net Services Administrator's Guide*
- *Oracle9i Real Application Clusters Concepts*
- *Oracle9i Database New Features*
- *Oracle9i Database Concepts*
- *Oracle9i Database Reference*
- *Oracle9i Database Error Messages*
- *Oracle9i Java Developer's Guide*
- *Oracle9i Java Stored Procedures Developer's Guide*

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 *Oracle9i Sample Schemas* for information on how these schemas were created and how you can use them yourself.

In North America, printed documentation is available for sale in the Oracle Store at

```
http://oraclestore.oracle.com/
```

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

```
http://www.oraclebookshop.com/
```

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

```
http://otn.oracle.com/admin/account/membership.html
```

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

```
http://otn.oracle.com/docs/index.htm
```

To access the database documentation search engine directly, please visit

```
http://tahiti.oracle.com
```

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples
- Conventions for Windows Operating Systems

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	DECIMAL (*digits* [, *precision*])
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	{ENABLE | DISABLE}
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	CREATE TABLE ... AS *subquery*; SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;

Convention	Meaning	Example
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME =` *`database_name`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Windows Operating Systems

The following table describes conventions for Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program. For example, to start Database Configuration Assistant, you must click the Start button on the taskbar and then choose Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.	Choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant
File and Directory Names	File and directory names are not case sensitive. The special characters <, >, :, ", /, |, and - are not allowed. The special character \ is treated as an element separator, even when it appears in quotes. If the file name begins with \ \, Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32`

Convention	Meaning	Example
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is "^". Your prompt reflects the subdirectory in which you are working. Referred to as the command prompt in this manual.	`C:\oracle\oradata>`

Special characters	The backslash special character (\) is sometimes required as an escape character for the double quote (") special character at the Windows command prompt. Parentheses and the single quote special character (') do not require an escape character. See your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"`
		`C:\>imp SYSTEM/`*`password`*
		`FROMUSER=scott TABLES=(emp, dept)`

HOME_NAME	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default was: ▪ `C:\orant` for Windows NT ▪ `C:\orawin98` for Windows 98 or whatever you called your Oracle home. This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install the latest Oracle release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora`*nn* where *nn* is the latest release number. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this manual follow OFA conventions. See *Oracle9i Database Getting Started for Windows* for additional information on OFA compliance and for information on installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\admin` directory.

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. Standards will continue to evolve over time, and Oracle Corporation 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 additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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 Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

What's New in Oracle COM Automation Feature?

The following sections describe the new features in Oracle COM Automation Feature:

- New Features in Oracle COM Automation Feature for Release 9.2
- New Features in Oracle COM Automation Feature for Release 9.0.1
- New Features in Oracle COM Automation Feature for Release 8.1.6

New Features in Oracle COM Automation Feature for Release 9.2

This section contains these topics:

- **Granting Privileges for Java**

  A new SQL script, `grant.sql`, must be run for each user of Oracle COM Automation for Java. The script grants the necessary privileges. For more information, see "Configuring Oracle COM Automation Feature for Java" on page 2-4.

- Many new features were added for Oracle9*i* release 1 (9.0.1). If you are upgrading from a pre-9.0.1 release, read the new features for "New Features in Oracle COM Automation Feature for Release 9.0.1".

New Features in Oracle COM Automation Feature for Release 9.0.1

This section contains these topics:

- **Using Oracle9*i* on Windows 2000**

  There are some differences between using Oracle9*i* on Windows 2000 and Windows NT 4.0.

  See Also: *Oracle9i Database Getting Started for Windows*

- **Sample Schema**

  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 *Oracle9i Sample Schemas* for information on how these schemas were created and how you can use them yourself.

- **New functionality to support Java Stored Procedures**

  Oracle COM Automation Feature is now available for Java as well as PL/SQL. While the general functionality is parallel, this document indicates those areas where functionality, setup, and architecture differ.

- **Migration from Oracle8i to Oracle9i**

  For this release, Oracle has renamed the `com81.dll` to `orawpcom.dll`. If you are migrating from Oracle8*i*, you must re-run `comwrap.sql` to continue using Oracle COM Automation Feature for PL/SQL.

- **Microsoft Word Java Demo**

  For Oracle COM Automation Feature for Java, a demo is available that shows how to load and run a Microsoft Word code sample.

- **Oracle COM Automation Feature for PL/SQL**

  Additional instructions have been presented to aid in use of the MAPI demo.

  See Also: "Setting Up the Environment to Use the MAPI Demo" on page 4-15 and "Preparing to Install MAPI Demo" on page 4-16.

- **Support provided for PL/SQL OUT parameters**

  `SetPtrArg` is being replaced with an implementation that provides the same functionality with `IN` and `OUT` parameters. This implementation consists of modification of `SetArg` functionality and `SetArg` datatype strings and the introduction of **InitOutArg** and **GetArg.**

  To use `OUT` parameters, see `SetArg` datatype strings in the section on "SetArg" on page 3-19.

New Features in Oracle COM Automation Feature for Release 8.1.6

Oracle8*i* release 8.1.6 included the following:

- **Two new demonstration programs**:
  - Microsoft PowerPoint Demo - Exchanges data from Oracle to PowerPoint.
  - MAPI Demo - Exchanges data from Oracle to Messaging Application Programming Interface (MAPI) compliant applications.

  See Also: Chapter 4, "Oracle COM Automation PL/SQL Demos" for information on using the demonstration programs.

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:

- Introduction to Oracle COM Automation Feature
- Benefits of Oracle COM Automation Feature
- Oracle COM Automation Feature Architecture

Introduction to 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 server on Windows NT, Windows 2000, and Windows XP.

You can build your own custom components or use the thousands of pre-built 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. It acts as a generic wrapper around the `IDispatch` interface.

- The feature externalizes all the 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:
  - Get the value of an exposed property.
  - Set the value of an exposed property.
  - Invoke a method on an object.

When an Oracle COM Automation Feature API is invoked from **PL/SQL** or Java Stored Procedures, the feature converts the parameters to the appropriate COM Automation datatypes 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 datatypes 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 server 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 NT, Windows 2000, and Windows XP. It provides these compelling benefits:

- 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 will have no problems incorporating these APIs into your **PL/SQL** subprograms or Java programs.

- Reusability

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

- 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 in a combined Oracle and Windows environment. You can be assured that Oracle integrates fully with and capitalizes on the services that are exposed by Windows NT, Windows 2000, Windows XP, Microsoft BackOffice applications, and Microsoft Office applications.

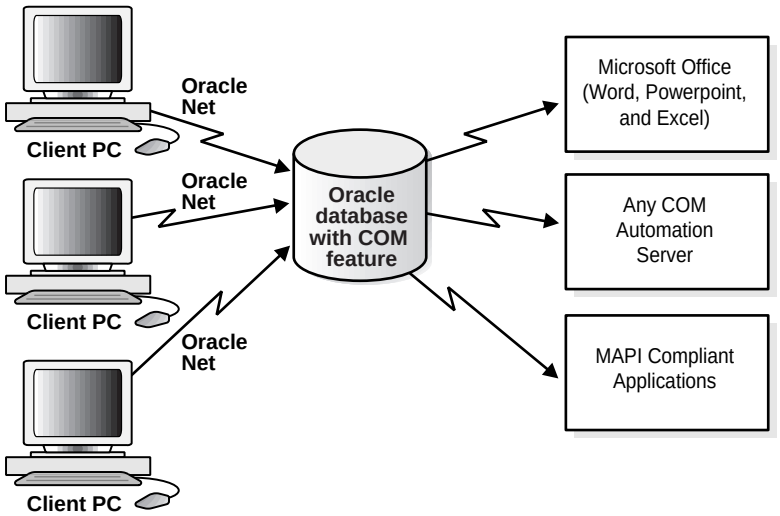
- Portability and Platform-Specific Requirements

  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 NT, Windows 2000, or Windows XP.

Oracle COM Automation Feature Architecture

Figure 1–1, "Oracle COM Interaction" 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.

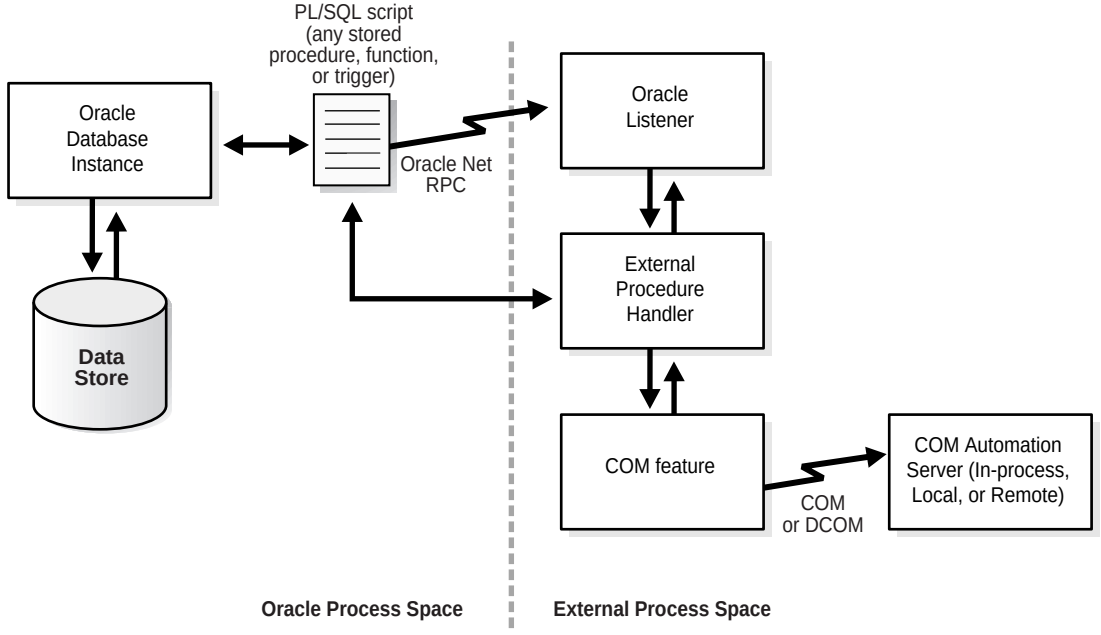
PL/SQL Architecture

Oracle COM Automation Feature for **PL/SQL** provides a package of PL/SQL APIs for manipulating COM objects. These APIs are implemented as **external procedures** 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 C functions that reside within a **DLL**. You can invoke Oracle COM Automation Feature APIs in the same manner as if you are calling a standard PL/SQL stored procedure or function.

Figure 1–2, "COM Automation Feature Architecture for PL/SQL" 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.dll`).
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. `extproc.exe` loads the **DLL** and executes the **external procedure**. Each of the COM Automation external procedure APIs in turn call Win32 APIs that

instantiate a COM object, set or get properties of a COM object, or invoke a method of a COM object.

5. `extproc.exe` 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 crash unexpectedly.

Java Architecture

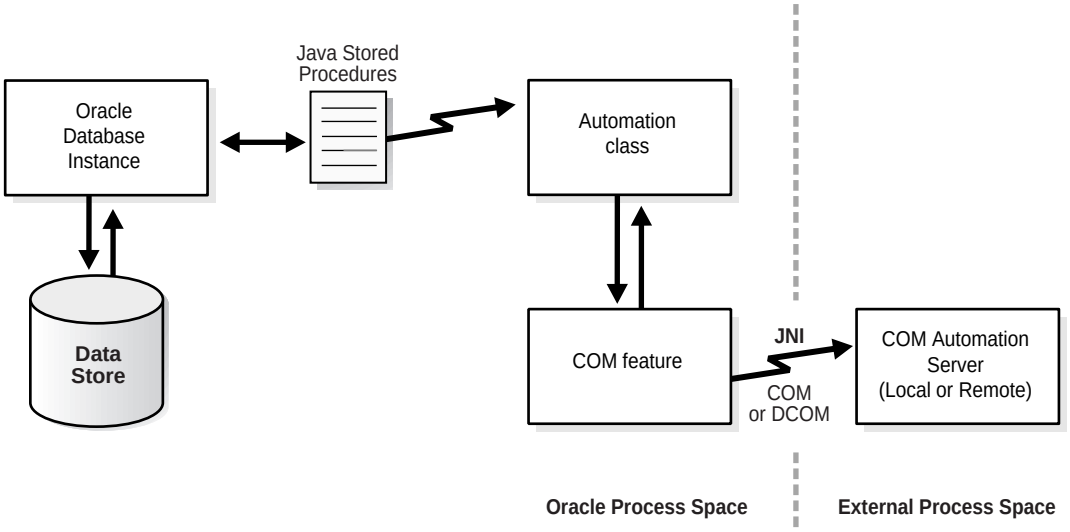
Oracle COM Automation Feature for Java is implemented by the Java Native Interface (JNI) as shown in Figure 1–3, "COM Automation Feature Architecture for Java".

The key components of this architecture are the `Automation` class and the Java COM Proxy DLL, `orawcom.dll`.

The interface is the `Automation` class, a Java proxy to 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.dll`, enables Java functions to invoke their corresponding COM functions.

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 installation and post-installation configuration tasks.

This chapter contains these topics:

- Oracle COM Automation Feature Components
- System Requirements
- Migration from Oracle8i to Oracle9i
- Configurations for Oracle COM Automation Feature
- Configuring the Listener for PL/SQL
- Support for DCOM

Oracle COM Automation Feature Components

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

> **See Also:** The *Oracle9i Database Installation Guide for Windows* for installation instructions

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

PL/SQL Components

- Oracle COM Automation PL/SQL feature (`orawpcom.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.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 JAR file, `orawcom.jar`
- Oracle COM Automation Java feature (`orawcom.dll`)
- Oracle COM Automation demonstration programs

Oracle COM Automation Java feature `orawcom.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

Note the following system requirements.

- Oracle COM Automation Feature requires:
  - Windows NT 4.0, Windows 2000, or Windows XP
  - a functioning Oracle database server on the computer before installation takes place
- For PL/SQL, Oracle COM Automation Feature requires:
  - Oracle8*i* or higher
- For Java, Oracle COM Automation Feature requires:
  - Oracle9*i* or higher
- Oracle COM Automation Feature demonstrations require that you first install the applications that are used in the demonstration programs.
  - The Word demos for PL/SQL and Java require Microsoft Word 95 or higher.
  - The Excel demo requires Microsoft Excel 95 or higher.
  - The PowerPoint demo requires Microsoft PowerPoint 97 or higher.
  - The MAPI demo requires Microsoft Outlook 2000 or higher.

  The demonstrations and installations are discussed in "Overview of PL/SQL Demos" on page 4-2 and "Oracle COM Automation Feature Java Demos Overview" on page 5-2.

Migration from Oracle8*i* to Oracle9*i*

For this release, Oracle has renamed `com81.dll` to `orawpcom.dll`. If you are migrating from Oracle 8*i*, you must re-run `comwrap.sql` to continue using Oracle COM Automation Feature for PL/SQL.

> **See Also:** "Configuring Oracle COM Automation Feature for PL/SQL" on page 2-4 for information on re-running `comwrap.sql`

Configurations for Oracle COM Automation Feature

Configuration procedures differ for **PL/SQL** and 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/password@net_service_name
   ```

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

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

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

   ```
   SQL> CONNECT hr/hr;
   SQL> @ORACLE_BASE\ORACLE_HOME\com\comwrap.sql
   ```

   where *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 normal.

Configuring Oracle COM Automation Feature for Java

To configure Oracle COM Automation Feature for Java:

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

   ```
   SQL> CONNECT SYSTEM/password@net_service_name
   ```

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

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

3. Run `loadjava` on the command prompt as follows:

   ```
   loadjava -force -resolve -user hr/hr
   ```

*ORACLE_BASE**ORACLE_HOME*\com\java\orawcom.jar

where hr is the user that uses Oracle COM Automation Feature.

> **See Also:** *Oracle9i Java Developer's Guide* for further information on the `loadjava` utility

Configuring the Listener for PL/SQL

When using Oracle COM Automation Feature for PL/SQL, there are specific requirements for the **listener.ora** and **tnsnames.ora** files that are described in this section.

For Oracle COM Automation Feature for Java, no special modifications to these files are required.

Because Oracle COM Automation Feature for PL/SQL relies on **external procedure** 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 inter-process communication (IPC) to invoke external stored procedures.

> **See Also:** *Oracle9i Net Services Administrator's Guide* for additional information on configuring the `listener.ora` and `tnsnames.ora` files for **external procedures**

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 > Control Panel.

   The Control Panel window appears.

2. Double-click Services.

   The Services dialog box appears.

3. Find `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 to start an Oracle service at the command prompt:

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

where *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 2-5 for information on how to configure these files

Support for DCOM

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

In order 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, "Services That Determine Security Credentials" 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

In order 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 > 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. Click the This Account radio button.
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 on:

- 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:

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

Datatype Conversions

Because Microsoft **COM** Automation uses COM Automation datatypes, and Oracle COM Automation Feature uses either PL/SQL or Java datatypes, Oracle COM Automation Feature must convert the data that it receives and pass it to the COM Automation object, and vice versa.

Datatype Conversion for PL/SQL

Table 3–1, "PL/SQL to COM Automation Datatypes" shows the mappings between PL/SQL datatypes and COM Automation datatypes.

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

Table 3–1 PL/SQL to COM Automation Datatypes

PL/SQL Datatype	COM Automation Datatype
VARCHAR2	BSTR, pBSTR
BOOLEAN	BOOL, pBOOL
BINARY_INTEGER	DISPATCH, pDISPATCH
DOUBLE PRECISION	UI1, pUI1, I2, pI2, I4, pI4, R4, pR4, R8, pR8, SCODE, pSCODE, CY, pCY, DISPATCH, pDISPATCH
DATE	DATE, pDATE

Note: Oracle restricts `CY` and `pCY` to this value: -9999999999.9999 to 9999999999.9999.

Datatype Conversion for Java

Table 3–2, "Java to COM Automation Datatypes" lists the supported COM Automation datatypes and related mappings to Java datatypes.

All the datatype mappings and return values apply to properties, arguments, and return values, except `void`, which only applies to return values.

Table 3–2 Java to COM Automation Datatypes

Java Datatype	COM Automation Datatype
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 on the HRESULT return code, refer to the Microsoft documentation.

> **See Also:** "Microsoft COM Automation Errors" on page A-4 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 HRESULT when an error occurs.

See Also: "GetLastError" on page 3-12 for additional information on 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 3-4

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 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 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.

Table 3–3 COMException Data Members

Member	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 helpfile.
`excepInfo`	if true, then `HRESULT` has the value `DISP_E_EXCEPTION`, and `source`, `description`, `helpfile`, and `helpid` contain more information.

Code Sample

This example demonstrates the `COMException` exception.

```
try
  {
    // Some code which 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 datatype 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 datatype, such as `INT` or `BSTR`. The `GetProperty` and `SetProperty` APIs take the property name and a variable of the appropriate datatype. Methods are referred to by a descriptive string, such as `InsertChart`. A method takes a set of parameters that are of different COM Automation datatypes and returns a COM Automation datatype.

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

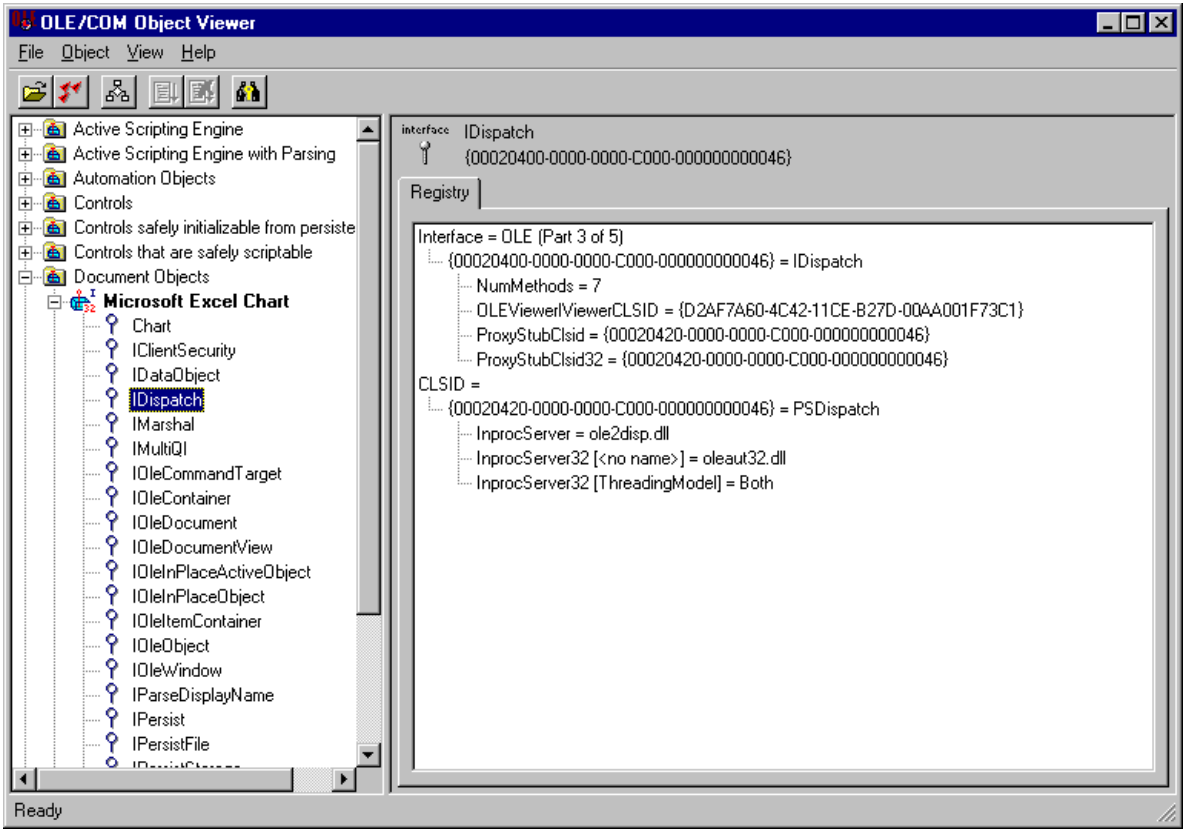
  ```
  [id(0x6003000)]
  long Post([in, out] long* lngAccountNo,
            [in, out] long* lngAmount,
            [in, out] BSTR* strResult);
  ```

  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, "OLE/COM Object Viewer" 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.

   As part of preparation for the `Invoke` API call, you use `InitArg` and `SetArg` in Java and you use `InitArg` and `SetArg` in PL/SQL 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

The feature externalizes the following APIs for PL/SQL development:

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

Java APIs

The 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

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: *COMComponent.Object* *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.

Where	Is
servername	the name of the remote **DCOM** server on which the COM object is being instantiated. 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 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 datatype 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 the interfaces associated with the object.

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

Code Sample

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

HRESULT :=ORDCOM.CreateObject('Excel.Application', 0, ‘’, applicationToken);
IF HRESULT = -1 THEN
  dbms_output.put_line(HRESULT);
END IF;
```

DestroyObject

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 the interfaces associated with the object.

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

Code Sample

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

/*
 At some point before this, we called CreateObject and
 got a valid applicationToken.
*/
HRESULT:=ORDCOM.DestroyObject(applicationToken);
```

GetLastError

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

See "Microsoft COM Automation Errors" on page A-4 for a description of the types of errors that can be returned by this function.

Code Sample

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

```
  return HRESULT;
END IF;
```

GetProperty

Gets a property value of a COM Automation object.

Syntax

```
FUNCTION GetProperty(objecttoken BINARY_INTEGER, propertyname VARCHAR2, argcount
BINARY_INTEGER, propertyvalue OUT any_PL/SQL_datatype) 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`.
`propertyvalue`	the returned property value. The returned property type depends on the COM Automation datatype that is returned. You must pass the PL/SQL datatype that corresponds to the COM Automation datatype of the COM Automation property. Otherwise, the COM Automation Feature will not properly convert the COM Automation datatype.
`any_PL/SQL_datatype`	any datatype supported by COM Automation Feature.

Remarks

If the property returns a COM object, you must specify a local variable of datatype `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, it is set to `NULL`.

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

Code Sample

```
ChartObject  BINARY_INTEGER:=-1;
ChartToken    BINARY_INTEGER := -1
HRESULT           BINARY_INTEGER;
/* Previously, ChartObject was initialized calling CreateObject */
HRESULT:=ORDCOM.getProperty(ChartObject, 'Chart', 0, ChartToken);
IF HRESULT=-1 THEN
  /* Do error checking here */
  return HRESULT;
END IF;
```

SetProperty

Sets a property of a COM Automation object to a new value.

Syntax

FUNCTION SetProperty(objecttoken BINARY_INTEGER, propertyname VARCHAR2, newvalue *any_PL/SQL_datatype*, datatype 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 datatype.
any_PL/SQL_datatype	any datatype supported by COM Automation Feature.

Where	Is
datatype	the explicitly specified datatype of the value passed in. The available datatypes are: ■ UI1 - byte integer ■ I2 - 2 byte integer ■ I4 - 4 byte integer ■ R4 - IEEE 4 byte real ■ R8 - IEEE 8 byte real ■ SCODE - error code ■ CY - currency (value - 9999999999.9999 to 9999999999.9999) (This is an Oracle restriction) ■ DISPATCH - dispatch pointer ■ BSTR - String ■ BOOL - boolean ■ DATE - date

Remarks

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

Code Sample

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

/*
 Previously, RangeToken has been initialized to a valid object token with a
property by the name of value.
*/
HRESULT:=ORDCOM.SetProperty(RangeToken, 'Value', ‘EmpNo’, ‘BSTR’);
IF HRESULT=-1 THEN
  /* Do error checking here */
  return HRESULT;
END IF;
```

InitArg

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 3-21 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` datatype strings in "SetArg" on page 3-30 for information about `IN` and `OUT` parameters.

Code Sample

See "Invoke" on page 3-31 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_datatype, type VARCHAR2);
```

Where	Is
data	the value of the OUT or IN OUT parameter after the COM method has been invoked.
any_PL/SQL_datatype	any datatype supported by COM Automation Feature.
datatype	the COM Automation datatype of the parameter. The available datatypes are: ■ pUI1 - byte integer ■ pI2 - 2 byte integer ■ pI4 - 4 byte integer ■ pR4 - IEEE 4 byte real ■ pR8 - IEEE 8 byte real ■ pSCODE - error code ■ pCY - currency (value -9999999999.9999 to 9999999999.9999) (This is an Oracle restriction) ■ pDISPATCH - dispatch pointer ■ pBSTR - String ■ pBOOL - boolean ■ pDATE - date

Remarks

See the section on SetArg datatype strings in "SetArg" on page 3-30 for information about IN and OUT parameters.

Code Sample

See "Invoke" on page 3-31 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_datatype, datatype 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 numbers of times `SetArg` has been called after an `InitArg` call.
`datatype`	the explicitly specified datatype for the parameters. Those datatypes 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 datatype. Those datatypes without the initial `p` are `IN` parameters. The available datatypes are: ■ `UI1` - byte integer ■ `pUI1` - byte integer ■ `I2` - 2 byte integer ■ `pI2` - 2 byte integer ■ `I4` - 4 byte integer ■ `pI4` - 4 byte integer ■ `R4` - IEEE 4 byte real ■ `pR4` - IEEE 4 byte real ■ `R8` - IEEE 8 byte real ■ `pR8` - IEEE 8 byte real ■ `SCODE` - error code ■ `pSCODE` - error code

Where	Is
	■ CY - currency (value -9999999999.9999 to 9999999999.9999) (This is an Oracle restriction) ■ pCY - currency (value -9999999999.9999 to 9999999999.9999) (This is an Oracle restriction) ■ DISPATCH - dispatch pointer ■ pDISPATCH - dispatch pointer ■ BSTR - String ■ pBSTR - String ■ BOOL - boolean ■ pBOOL - boolean ■ DATE - date ■ pDATE - date
any_PL/SQL_datatype	any datatype 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.

Datatypes without the initial *p* are IN parameters. Those datatypes prefaced by an initial *p* are IN OUT or OUT parameters.

Code Sample

See "Invoke" on page 3-21 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

```
FUNCTION Invoke(objecttoken BINARY_INTEGER, methodname VARCHAR2, argcount
BINARY_INTEGER, returnvalue OUT any_PL/SQL_datatype) 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 datatype.
`any_PL/SQL_datatype`	any datatype supported by COM Automation Feature.

Remarks

If the method's return value is a COM object, then the developer must specify a local variable of datatype `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 procedure returns a `0` when successful or a nonzero `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);
```

```
 */

applicationToken binary_integer:=-1;
i 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');

i:=ORDCOM.Invoke(applicationToken, 'TestOutArg', 3, x4);

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 datatype. CURRENCY is a an 8-byte number where the last 4 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

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: *`COMComponent.Object`* *`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. 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 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.

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 the Oracle COM Automation Feature and releases all the 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 exception `COMException` 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 machine.
  Automation excel = new Automation("Excel.Application",
                                    "\\ServerMachineName");

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

Create

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: *`COMComponent.Object`* *`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.

Where	Is
	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 the Oracle COM Automation Feature and releases all the 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 exception `COMException` 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 machine.
 excel.Create("Excel.Application", "\\ServerMachineName");

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

Destroy

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 the interfaces associated with the object.

Code Sample

See "Create" on page 3-25 for code sample.

GetProperty

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 in order to return the property.
`allowed_type`	from the following list: ▪ `boolean` ▪ `byte` ▪ `char` ▪ `short` ▪ `int` ▪ `long` ▪ `float` ▪ `double` ▪ `java.long.String` ▪ `oracle.win.com.Automation` ▪ `oracle.win.com.Currency` ▪ `java.util.Calendar`

Remarks

If the property is a COM object, 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 in order to overload the `GetProperty` method. Overloading would not be possible if the property value were simply 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 exception, `COMException`, 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

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.

Where	Is
`propVal`	the new value of the property. It must be a value of the appropriate datatype.
`allowed_type`	from the following list: ■ `boolean` ■ `byte` ■ `char` ■ `short` ■ `int` ■ `long` ■ `float` ■ `double` ■ `java.long.String` ■ `oracle.win.com.Automation` ■ `oracle.win.com.Currency` ■ `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 exception, `COMException`, is thrown if an error occurs.

Code Sample

See `"GetProperty"` on page 3-27 for sample code.

InitArg

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 3-31 for sample code.

SetArg

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 numbers of times `SetArg` has been called after an `InitArg` call.
`allowed_type`	from the following list. ▪ `boolean` ▪ `byte` ▪ `char` ▪ `short` ▪ `int` ▪ `long` ▪ `float` ▪ `double`

Where	Is
	▪ `java.long.String` ▪ `oracle.win.com.Automation` ▪ `oracle.win.com.Currency` ▪ `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 3-31 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, it must be a local variable of the appropriate datatype. The array must be big enough to hold at least one element, although only the first element will be accessed in order to return the property.
`allowed_type`	a type from the following list:

Where	Is
	■ `boolean` ■ `byte` ■ `char` ■ `short` ■ `int` ■ `long` ■ `float` ■ `double` ■ `java.long.String` ■ `oracle.win.com.Automation` ■ `oracle.win.com.Currency` ■ `java.util.Calendar`

Remarks

If the COM method returns a COM object as the return value, 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.

In order 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 simply 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 exception `COMException` 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 which takes no arguments.
workSheet.InitArg();
workSheet.Invoke("ChartObjects", autorv);
chartObjects = autorv[0];

// Invoke a method which 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

Creates a `currency` Java object.

Syntax

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

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

Get

Gets the 8-byte CURRENCY number.

Syntax

```
public long Get()
```

Remarks

Returns the 8-byte CURRENCY number.

Set

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 PL/SQL Demos
- Microsoft Word Demo
- Microsoft Excel Demo
- Microsoft PowerPoint Demo
- MAPI Demo

Overview of 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, Oracle Corporation has provided the following demos:

- Microsoft Word Demo - Exchanges data from Oracle to Word
- Microsoft Excel Demo - Exchanges data from Oracle to Excel
- Microsoft PowerPoint Demo - Exchanges data from Oracle to PowerPoint
- MAPI Demo - Exchanges data from Oracle to Messaging Application Programming Interface (MAPI) compliant applications

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 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 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 installing this demo.

To install the Microsoft Word demos:

1. Start SQL*Plus.

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

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

   ```
   SQL> connect hr/hr
   ```

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 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 `worddemo.doc` to see its contents.

Core Functionality

The following subsections describe the APIs that the Microsoft Word demo exposes. These APIs are primitive. Be aware that much of the functionality that Microsoft Word exposes through COM Automation is not exposed through these APIs.

CreateWordObject

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

FileNew

Creates a new Microsoft Word document.

Syntax

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

Remarks

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

FileLoad

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

FileSave

Saves the current Microsoft Word document to disk.

Syntax

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

Remarks

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

FileSaveAs

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

FileClose

Closes the current Microsoft Word document.

Syntax

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

Remarks

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

InsertText

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

InsertNewLine

Inserts a carriage return into the current Microsoft Word document.

Syntax

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

Remarks

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

FormatFontSize

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 procedure returns a 0 when successful or a nonzero 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 Word 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 and Microsoft Word. It exchanges data from the `EMPLOYEES` and `JOBS` tables in Oracle 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 installing 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 that will use the Microsoft Excel demo. For example:

   ```
   SQL> connect hr/hr
   ```

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 current user's schema. You 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 its contents.

Core Functionality

The following subsections describe the APIs that the Microsoft Excel demo exposes. These APIs are primitive. Be aware that much of the functionality that Microsoft Excel exposes through COM Automation is not exposed through these APIs.

CreateExcelWorkSheet

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

InsertData

Inserts any kind of data into a specific cell of the current Excel worksheet.

Syntax

```
FUNCTION InsertData(range VARCHAR2, data ANY PL/SQL DATATYPE, datatype 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.
datatype	a string that indicates the datatype of the data that you are inserting into Excel. The list of available datatypes are: ■ I2 - 2 byte integer ■ I4 - 4 byte integer ■ R4 - IEEE 4 byte real ■ R8 - IEEE 8 byte real ■ SCODE - error code ■ CY - currency ■ DISPATCH - dispatch pointer ■ BSTR - String ■ BOOL - boolean ■ DATE - date

Remarks

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

InsertChart

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 datatype of the data to be graphed.

Remarks

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

SaveExcelFile

Saves the current Microsoft Excel workbook as a specific file.

Syntax

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

Where	Is
filename	the fully qualified filename of the Excel workbook

Remarks

This procedure returns a 0 when successful or a nonzero 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 procedure 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 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 and Microsoft PowerPoint. It exchanges data from the `EMPLOYEES` and `JOBS` tables in Oracle 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 97 or later 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 that will use the Microsoft PowerPoint demo. For example:

   ```
   SQL> connect hr/hr
   ```

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 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. Be aware that much of the functionality that Microsoft PowerPoint exposes through COM Automation is not exposed through these APIs.

CreatePresentation

Starts the Microsoft PowerPoint COM Automation server and instantiates the objects for a presentation.

Syntax

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

Remarks

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

AddSlide

Inserts a new slide in the PowerPoint presentation.

Syntax

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

Remarks

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

SetTitle

Specifies the title of the PowerPoint slide.

Syntax

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

Remarks

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

InsertText

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;
```

Remarks

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

PresentationSave

Saves the current PowerPoint presentation.

Syntax

```
FUNCTION PresentationSave RETURN BINARY_INTEGER;
```

Remarks

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

PresentationSaveAs

Saves the current presentation using the specified name.

Syntax

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

Remarks

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

PresentationClose

Closes the current PowerPoint presentation.

Syntax

```
FUNCTION PresentationClose RETURN binary_integer;
```

Remarks

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

Exit

Exits the PowerPoint program.

Syntax

```
FUNCTION Exit RETURN BINARY_INTEGER;
```

Remarks

This procedure returns a `0` when successful or a nonzero `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 Oracle and MAPI-compliant applications.

The MAPI demo provides the following:

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

Setting Up the Environment to Use the MAPI Demo

You must set up certain related applications in order 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:

   Start > Programs > Microsoft Exchange > Active Directory Users and Computers.

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

2. Install Microsoft Outlook as follows:

   Select custom install. Select the Collaboration Data Objects (these are not installed by default).

   During the install, select the Corporate or Workgroup option.

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

   Add the account 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 calender tools as follows:

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

Preparing to Install MAPI Demo

The **MAPI** Solution invokes Extended MAPI client on the behalf of the Oracle database server. The Oracle database service on Windows NT, by default, runs as NT system user `LocalSystem`. The MAPI profile for user `LocalSystem` is not easily configured. Before using the MAPI Solution, change both the Windows NT Oracle database service and `Oracle`*`HOME_NAME`*`TNSListener` service to start up using a Windows NT login user account.

To prepare to install the MAPI demo:

1. Log on to Windows NT 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 NT user `DOMAIN-1\hr`. Make sure that you are able to send out e-mail using this profile.
3. Go to the Windows NT 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 the 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 installing this demo.

To install the MAPI demo:

1. Start SQL*Plus.

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

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

   ```
   SQL> connect hr/hr
   ```

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 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. Save the change.

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. Be aware that much of the functionality that MAPI exposes through COM Automation is not exposed through these APIs.

CreateMAPISession

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;
```

Remarks

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

NewMessage

Creates a new message.

Syntax

```
FUNCTION NewMessage RETURN BINARY_INTEGER;
```

Remarks

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

AddRecipient

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;
```

Remarks

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

SetSubject

Specifies the subject of the e-mail message.

Syntax

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

Remarks

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

SetBody

Inserts the body text of the e-mail message.

Syntax

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

Remarks

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

SendMessage

Sends the e-mail message to the specified recipients.

Syntax

```
FUNCTION SendMessage RETURN BINARY_INTEGER;
```

Remarks

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

EndMAPISession

Exits the MAPI session.

Syntax

```
FUNCTION EndMAPISession RETURN BINARY_INTEGER;
```

Remarks

This procedure returns a `0` when successful or a nonzero `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:

- Oracle COM Automation Feature Java Demos Overview
- Microsoft Word Java Demo

Oracle COM Automation Feature Java Demos Overview

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 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, Oracle Corporation has provided the following demo:

- Microsoft Word Java Demo - Exchanges data from Oracle to 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 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 which creates the call spec for the demo.

Installing the Microsoft Word Java Demo

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

To install the demo:

1. Run `loadjava` from the command line:

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

2. Start SQL*Plus.

   ```
   C:\>SQLPLUS
   ```

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

   ```
   SQL> connect hr/hr
   ```

4. Run `TestWORD.sql` to create the call spec:

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

See Also:

- *Oracle9i Java Developer's Guide* for further information on using the `loadjava` tool
- *Oracle9i Java Stored Procedures Developer's Guide* for further information on call specs

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 `wordemoj.doc` to see its contents.

Creating a Custom Application

`public class TestWORD` as described in "Core Functionality" on page 5-4, provides a wrapper around the Word.Basic COM Automation class as well as some sample code which demonstrates how to use the wrapper. This code was written to be run on the Oracle database server.

To create a custom application which uses this wrapper:

1. Instantiate an object of this class.
2. Create the Word.Basic object by calling the method `CreateWordObject`.
3. Create a new Microsoft Word document with the method `FileNew`, or open an existing document with the method `FileLoad`.
4. Use the methods `FormatFontSize`, `InsertText`, `InsertNewLine` to add text and formatting to the document.
5. Save the document with `FileSaveAs` or `FileSave`.
6. Call the method `FileClose` when you are finished with the document.
7. Call the method `DestroyWordObject` 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. Be aware that much of the functionality that

Microsoft Word exposes through COM Automation is not exposed through these APIs.

TestWORD

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

Destroys the Word.Basic COM object.

Syntax

```
public void DestroyWordObject()
```

FileNew

Creates a new Microsoft WORD document.

Syntax

```
public void FileNew()
```

Remarks

Wrapper for the `FileNewDefault` COM method of the Word.Basic COM object.

FileLoad

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

Wrapper for the `FileOpen` COM method of the Word.Basic COM object.

FormatFontSize

Sets the font size.

Syntax

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

Where	Is
fontsize	the new font size.

Remarks

Wrapper for the `FormatFont` COM method of the Word.Basic COM object.

InsertText

Inserts text into the Microsoft Word document.

Syntax

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

Where	Is
textstr	the text to insert.

Remarks

Wrapper for the `Insert` COM method of the Word.Basic COM object.

InsertNewLine

Inserts a newline into the Microsoft Word document.

Syntax

```
public void InsertNewLine()
```

Remarks

Wrapper for the `InsertPara` COM method of the Word.Basic COM object.

FileSaveAs

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

Wrapper for the `FileSaveAs` COM method of the Word.Basic COM object.

FileSave

Saves the Microsoft Word document.

Syntax

```
public void FileSave()
```

Remarks

Wrapper for the `FileSave` COM method of the Word.Basic COM object.

FileClose

Closes the Microsoft Word document, and exits Microsoft Word.

Syntax

```
public void FileClose()
```

Remarks

Wrapper for the `FileClose` and `FileExit` COM methods of the Word.Basic COM object.

A

COM Automation Error Messages

This appendix includes the following:

- Oracle COM Automation Feature, PL/SQL Errors
- Microsoft COM Automation Errors

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

Action: Make sure that the variable is of the appropriate datatype.

COM-0002: Invalid Token or no interface for token

Action: Make sure that the interface exists.

COM-0003: Maximum Objects reached

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

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

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

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

Action: Make sure your registry can be written to and is not corrupted.

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

Action: Make sure the class is registered.

COM-0007: Failed to initialize COM Automation object

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

COM-0008: No interface is supported

Action: Check that the interface specified is valid.

COM-0009: Failed to get type info count

Action: Check that the object is properly registered.

COM-0010: Does not support type info implementation

Action: Check that the object is properly registered.

COM-0011: Failed to get type information

Action: Check that the object is properly registered.

COM-0012: Failed to get type attributes

Action: Check that the object is properly registered.

COM-0013: Failed to get function description at index

Action: Check that the object is properly registered.

COM-0014: Failure to invoke

Action: Check that the method name is valid for the object.

COM-0015: Bad parameter count

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

COM-0016: One of the arguments in rgvarg is not a valid variant type

Action: Check that the object is properly registered.

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

Action: Make sure the structure in `pexcepinfo` is initialized.

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

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

COM-0019: This implementation of IDispatch does not support named arguments

Action: Do not use named arguments. Use standard parameter passing.

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

Action: Make sure that the coerced arguments are of compatible datatypes.

COM-0021: One of the parameter dispatch IDs does not correspond to a parameter on the method

Action: Make sure the arguments are passed in correctly.

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

Action: Make sure your arguments are compatible.

COM-0023: The interface ID passed in riid is not IID_NULL

Action: Make sure the interface ID passed is IID_NULL.

COM-0024: The member being invoked interprets string arguments according to an unrecognized locale ID (LCID)

Action: Make sure your `localeID` is valid.

COM-0025: Not an optional parameter

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

COM-0026: Name exceeded the maximum character allowed

Action: Enter 1024 characters or less for the name.

COM-0027: This class cannot be created as part of an aggregate

Action: Do not create this class as part of an aggregate.

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.

(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.

(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 datatype of a PL/SQL local variable used to store a returned property value or a method return value did not match the Visual Basic datatype of the property or method return value.

Action: Ensure that the local variable is of the appropriate datatype. Or, the return value of a property or a method was requested, but it does not return a value.

(0x80020006) (-2147352570) Unknown name

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

(0x80020008) (-2147352568) Bad variable type

Cause: The datatype of a PL/SQL or Java value passed as a method parameter did not match the COM Automation datatype 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 datatype 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.

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)

MAPI is 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 COM Automation interface (IDispatch).

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 Corporation'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

A

APIs
  Java, 3-22
  PL/SQL, 3-10
application programming interfaces (APIs), 3-9
  Java, 3-22
  PL/SQL, 3-10
Architecture
  Java, 1-7
  PL/SQL, 1-5
architecture
  Oracle COM Automation, 1-5
Automation
  Java API, 3-24, 3-33

B

benefits
  Oracle COM Automation, 1-3

C

call spec, 5-3
COM automation
  invoking, 1-6
  PL/SQL errors, A-2
COM objects
  program ID, 3-6
  properties and methods, 3-6
  required information, 3-6
  viewing, 3-7
comwrap.sql, 2-2, 2-4
configuration
  Java, 2-4
  PL/SQL, 2-4
configuring
  DCOM, 2-7
  listener for Oracle COM Automation for PL/SQL, 2-5
  Oracle COM Automation, 2-4
constructor, 3-24, 3-33
core functionality
  Oracle COM Automation, 1-2
Create
  Java API, 3-25
CreateObject
  PL/SQL API, 3-10

D

datatypes
  conversion, 3-2
  Java to COM Automation datatypes, 3-2
  PL/SQL to Visual Basic, 3-2
DCOM
  configuring, 2-7
demos
  installing MAPI demo, 4-17
  installing Microsoft Excel demo, 4-7
  installing Microsoft PowerPoint demo, 4-11
  installing Microsoft Word demo, 4-3
  MAPI, 4-15
  Microsoft Excel, 4-7
  Microsoft PowerPoint, 4-11
  Microsoft Word, 4-2
  Oracle COM Automation, 4-2

Oracle COM Automation for Java, 5-2
PL/SQL, 4-2
Destroy
Java API, 3-26
DestroyObject
PL/SQL API, 3-12

E

errors
codes, 3-3
HRESULT, 3-3
messages, A-4
Microsoft COM automation, A-4
Oracle COM automation, A-2
examples
MAPI, 4-15
Microsoft Excel, 4-7
Microsoft PowerPoint, 4-11
Microsoft Word, 4-2
Microsoft Word Java, 5-2
Exchange Server, 4-15
EXTPROC
extproc.exe, 1-6

G

GetArg
PL/SQL API, 3-18
GetLastError
PL/SQL API, 3-12
GetProperty
Java API, 3-27
PL/SQL API, 3-14
Globally Unique Identifier (GUID), 3-6
grant.sql, 2-4

H

HRESULT
return codes, 3-3

I

IDispatch interface, 3-10, 3-22, 3-24
IDL, 3-6
InitArg
Java API, 3-29
PL/SQL API, 3-17
InitOutArg
PL/SQL API, 3-17
installation
Oracle COM Automation, 2-2
installing PL/SQL MAPI demo
preparation, 4-16
Installing the Microsoft Word Java Demo, 5-3
Interface Definition Language, 3-6
Internet Explorer, 4-16
Invoke
Java API, 3-31
PL/SQL API, 3-21

J

Java
configuration, 2-4
Java API
Automation, 3-24, 3-33
Create, 3-25
Destroy, 3-26
GetProperty, 3-27
InitArg, 3-29
Invoke, 3-31
SetArg, 3-30
SetProperty, 3-28
Java APIs, 3-22
Java Automation constructor, 3-24, 3-33
Java Components, 2-2

L

listener
configuring for Oracle COM Automation, 2-5
loadjava, 5-3

M

MAPI
  demo script mapidem.sql, 4-15
  PL/SQL example, 4-15
MAPI demo, 4-16
Messaging Application Programming Interface. See MAPI
Microsoft Excel
  demo script exceldem.sql, 4-7
  PL/SQL example, 4-7
Microsoft Outlook, 4-16
Microsoft PowerPoint
  demo script pptdem.sql, 4-11
  PL/SQL example, 4-11
Microsoft Word
  demo script worddem.sql, 4-2, 4-7
  example, 4-2
Microsoft Word Java Demo, 5-2
Migration from Oracle8i to Oracle 9i, 2-3

O

OLE/COM Object Viewer, 3-7
ORA-28575 error message, 2-6
Oracle COM Automation
  architecture, 1-5
  benefits, 1-3
  components, 2-2
  configuring, 2-4
  core functionality, 1-2
  demos, 4-2
  installing, 2-2
  introduction, 1-2
  Java demos, 5-2
orawcom.dll, 2-2
orawpcom.dll, 1-6, 2-2
ORDExcel
  PL/SQL package, 4-7
ORDMAPI
  PL/SQL package, 4-15
ORDPPT
  PL/SQL package, 4-11
ORDWord
  PL/SQL package, 4-2
Outlook Client, 4-15

P

PL/SQL
  configuration, 2-4
  ORDExcel package, 4-7
  ORDMAPI package, 4-15
  ORDPPT package, 4-11
  ORDWord package, 4-2
PL/SQL API
  CreateObject, 3-10
  DestroyObject, 3-12
  GetArg, 3-18
  GetLastError, 3-12
  GetProperty, 3-14
  InitArg, 3-17
  InitOutArg, 3-17
  Invoke, 3-21
  SetArg, 3-19
  SetProperty, 3-15
PL/SQL Architecture, 1-5
PL/SQL Components, 2-2
preparing to install, 4-16
progID
  COM objects, 3-6
program ID
  COM objects, 3-6

R

return codes
  HRESULT, 3-3

S

sample schema, 4-2, 5-2
SERVEROUTPUT, 5-4
SetArg
  Java API, 3-30
  PL/SQL API, 3-19
SetProperty
  Java API, 3-28
  PL/SQL API, 3-15

system requirements
Oracle COM Automation, 2-3

T

TestWORD, 5-4
TestWORD.class, 5-2
TestWORD.java, 5-2
TestWORD.sql, 5-2, 5-3
troubleshooting
Oracle COM automation PL/SQL errors, A-2

U

Using the Microsoft Word Demo, 5-4

W

wordemoj.doc, 5-4