4

Advanced Interface Methods

Objectives

After completing this lesson, you should be able to do the following:

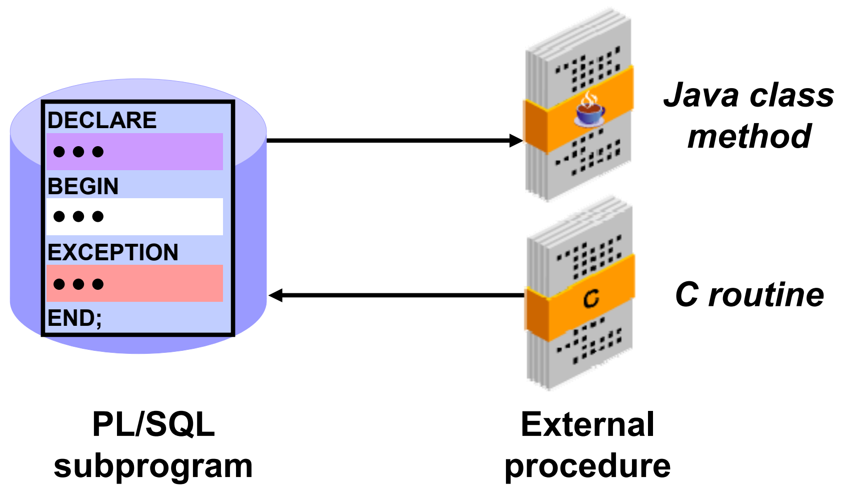
- Execute external C programs from PL/SQL
- Execute Java programs from PL/SQL

Objectives

In this lesson, you learn how to implement an external C routine from PL/SQL code and how to incorporate Java code into your PL/SQL programs.

Calling External Procedures from PL/SQL

With external procedures, you can make "callouts" and, optionally, "callbacks" through PL/SQL.



External Procedures: Overview

An *external procedure* (also called an *external routine*) is a routine stored in a dynamic link library (DLL), shared object (`.so` file in UNIX), or libunit in the case of a Java class method, that can perform special purpose processing. You publish the routine with the base language and then call it to perform special purpose processing. You call the external routine from within PL/SQL or SQL. With C, you publish the routine through a library schema object, which is called from PL/SQL, that contains the compiled library file name that is stored on the operating system. With Java, publishing the routine is accomplished through creating a class libunit.

A *callout* is a call to the external procedure from your PL/SQL code.

A *callback* occurs when the external procedure calls back to the database to perform SQL operations. If the external procedure is to execute SQL or PL/SQL, it must "call back" to the database server process to get this work done.

An external procedure enables you to:

- Move computation-bound programs from the client to the server where they execute faster (because they avoid the round trips entailed in across-network communication)
- Interface the database server with external systems and data sources
- Extend the functionality of the database itself

Benefits of External Procedures

- **External procedures integrate the strength and capability of different languages to give transparent access to these routines from within the database.**
- **Extensibility: Provides functionality in the database that is specific to a particular application, company, or technological area**
- **Reusability: Can be shared by all users on a database, as well as moved to other databases or computers, providing standard functionality with limited cost in development, maintenance, and deployment**

Benefits of External Procedures

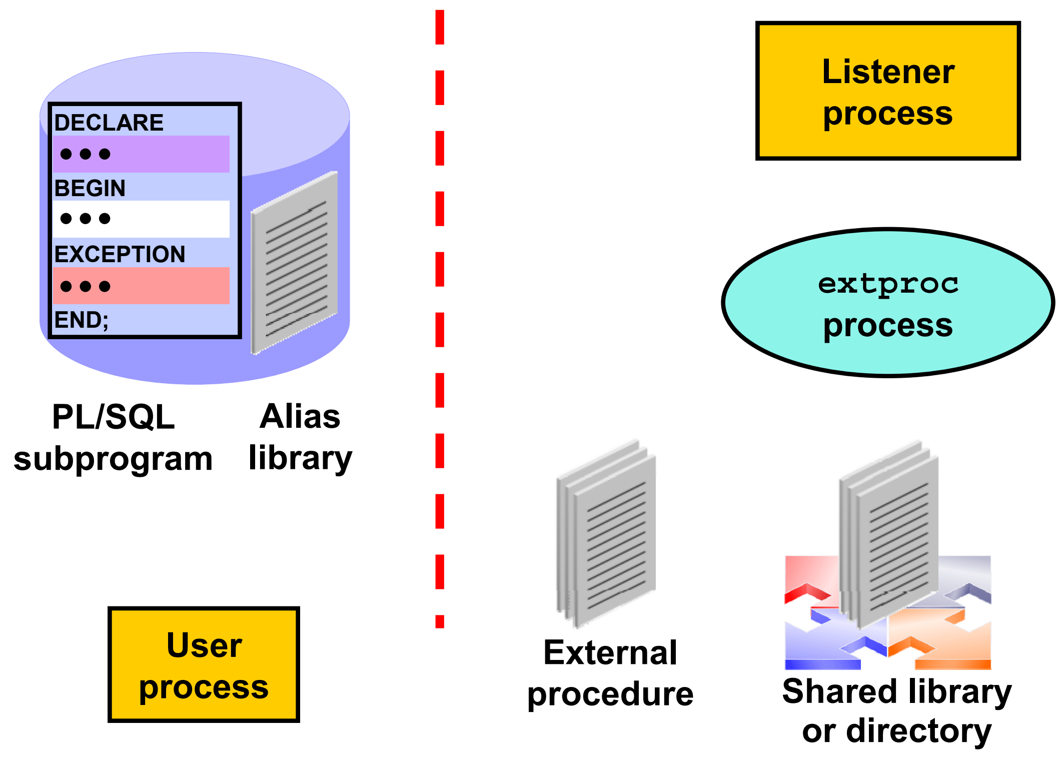
If you use the external procedure call, you can invoke an external routine by using a PL/SQL program unit. Additionally, you can integrate the powerful programming features of 3GLs with the ease of data access of SQL and PL/SQL commands.

You can extend the database and provide backward compatibility. For example, you can invoke different index or sorting mechanisms as an external procedure to implement data cartridges.

Example

A company has very complicated statistics programs written in C. The customer wants to access the data stored in an Oracle database and pass the data into the C programs. After the execution of the C programs, depending on the result of the evaluations, data is inserted into the appropriate Oracle database tables.

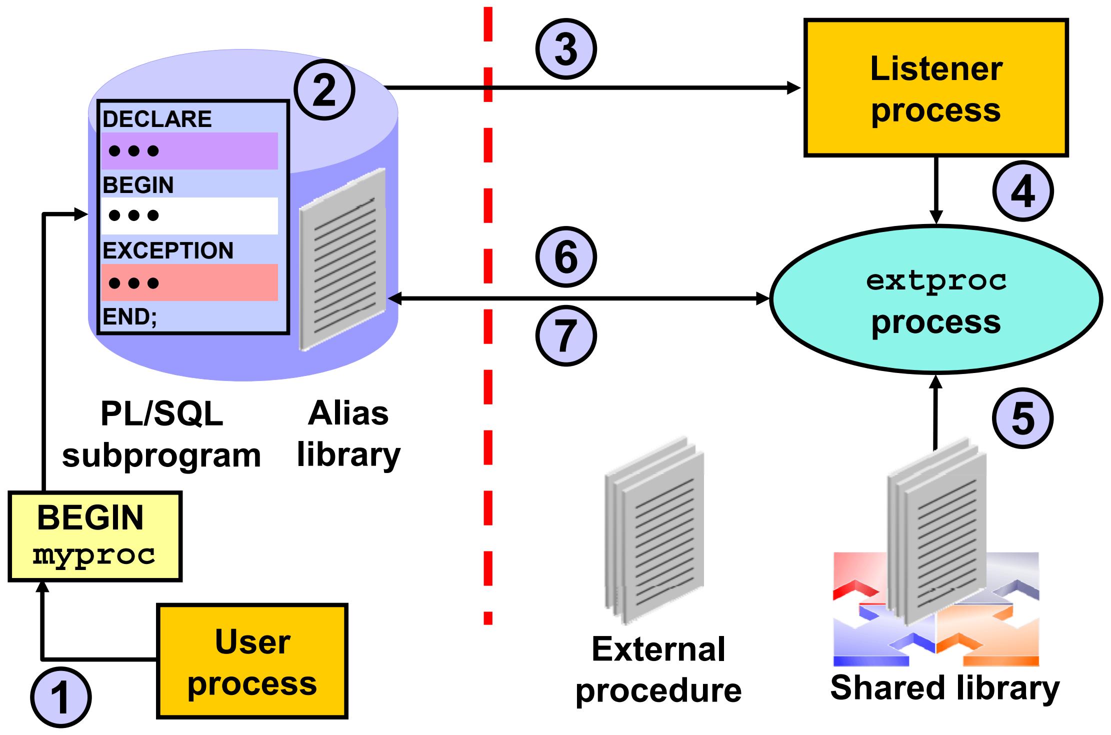
External C Procedure Components



External C Procedure Components

- **External procedure:** A unit of code written in C
- **Shared library:** An operating system file that stores the external procedure
- **Alias library:** A schema object that represents the operating system shared library
- **PL/SQL subprograms:** Packages, procedures, or functions that define the program unit specification and mapping to the PL/SQL library
- **`extproc` process:** A session-specific process that executes external procedures
- **Listener process:** A process that starts the `extproc` process and assigns it to the process executing the PL/SQL subprogram

How PL/SQL Calls a C External Procedure



How an External C Procedure Is Called

1. The user process invokes a PL/SQL program.
2. The server process executes a PL/SQL subprogram, which looks up the alias library.
3. The PL/SQL subprogram passes the request to the listener.
4. The listener process spawns the `extproc` process. The `extproc` process remains active throughout your Oracle session until you log off.
5. The `extproc` process loads the shared library.
6. The `extproc` process links the server to the external file and executes the external procedure.
7. The data and status are returned to the server.

The extproc Process

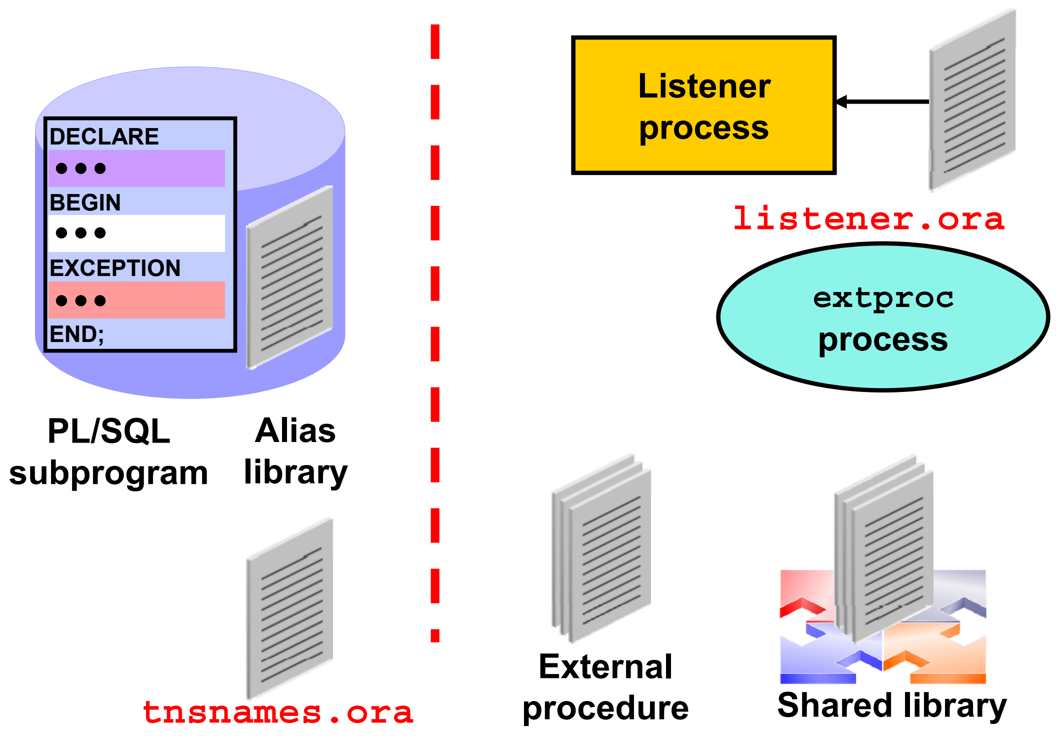
- The extproc process services the execution of external procedures for the duration of the session until the user logs off.
- Each session uses a different extproc process to execute external procedures.
- The listener must be configured to allow the server to be associated to the extproc process.
- The listener must be on the same machine as the server.

The extproc Process

The extproc process performs the following actions:

- **Converts PL/SQL calls to C calls:**
  - Loads the dynamic library
- **Executes the external procedures:**
  - Raises exceptions if necessary
  - Converts C back to PL/SQL
  - Sends arguments or exceptions back to the server process

The Listener Process



The Listener Process

When the Oracle server executes the external procedure, the request is passed to the listener process, which spawns an `extproc` process that executes the call to the external procedure.

This listener returns the information to the server process. A single `extproc` process is created for each session. The listener process starts the `extproc` process. The external procedure resides in a dynamic library. The Oracle Server 10*g* runs the `extproc` process to load the dynamic library and to execute the external procedure.

3GL Call Dependencies: Example

Libraries are objects with the following dependencies. Given library L1 and procedure P1, which depends on L1, when procedure P1 is executed, library L1 is loaded, and the corresponding external library is dynamically loaded. P1 can now use the external library handle and call the appropriate external functions.

If L1 is dropped, then P1 is invalidated and needs to be recompiled.

Development Steps for External C Procedures

1. Create and compile the external procedure in 3GL.
2. Link the external procedure with the shared library at the operating system level.
3. Create an alias library schema object to map to the operating system's shared library.
4. Grant execute privileges on the library.
5. Publish the external C procedure by creating the PL/SQL subprogram unit specification, which references the alias library.
6. Execute the PL/SQL subprogram that invokes the external procedure.

Development Steps for External C Procedures

Steps 1 and 2 will vary according to the operating system. Consult your operating system or the compiler documentation. After those steps are completed, you need to create an alias library schema object that identifies the operating system's shared library within the server. Any user who needs to execute the C procedure requires execute privileges on the library. Within your PL/SQL code, you map the C arguments to PL/SQL parameters, and lastly, execute your PL/SQL subprogram that invokes the external routine.

Development Steps for External C Procedures

1. *Varies for each operating system; consult documentation.*
2. Use the `CREATE LIBRARY` statement to create an alias library object.

```
CREATE OR REPLACE LIBRARY library_name IS|AS
'file_path';
```

3. Grant the `EXECUTE` privilege on the alias library.

```
GRANT EXECUTE ON library_name TO
user|ROLE|PUBLIC;
```

Create the Alias Library

An alias library is a database object that is used to map to an external shared library. Any external procedure that you want to use needs to be stored in a dynamic link library (DLL) or shared object library (SO) operating system file. The DBA controls access to the DLL or SO files by using the `CREATE LIBRARY` statement to create a schema object called an alias library, that represents the external file. The DBA needs to give you `EXECUTE` privileges on the library object so that you can publish the external procedure and then call it from a PL/SQL program.

Steps

1, 2. Steps 1 and 2 will vary for each operating system. Consult your operating system or the compiler documentation.

3. Create an alias library object by using the `CREATE LIBRARY` command:

```
CREATE OR REPLACE LIBRARY c_utility
AS '$ORACLE_HOME/bin/calc_tax.so';
```

The example shows the creation of a database object called `c_utility`, which references the location of the file and the name of the operating system file, `calc_tax.so`.

Create the Alias Library (continued)

4. Grant the `EXECUTE` privilege on the library object:

```
       SQL> GRANT EXECUTE ON c_utility TO OE;
```

5. Publish the external C routine.
6. Call the external C routine from PL/SQL.

Dictionary Information

The alias library definitions are stored in the `USER_LIBRARIES` and `ALL_LIBRARIES` data dictionary views.

Development Steps for External C Procedures

Publish the external procedure in PL/SQL through call specifications:

- **The body of the subprogram contains the external routine registration.**
- **The external procedure runs on the same machine.**
- **Access is controlled through the alias library.**

Method to Access a Shared Library Through PL/SQL

You can access a shared library by specifying the alias library in a PL/SQL subprogram. The PL/SQL subprogram then calls the alias library.

- The body of the subprogram contains the external procedure registration.
- The external procedure runs on the same machine.
- Access is controlled through the alias library.

You can publish the external procedure in PL/SQL by:

- Identifying the characteristics of the C procedure to the PL/SQL program
- Accessing the library through PL/SQL

The package specification does not require any changes. You do not need to have definitions for the external procedure.

The Call Specification

Call specifications enable:

- **Dispatching the appropriate C or Java target procedure**
- **Data type conversions**
- **Parameter mode mappings**
- **Automatic memory allocation and cleanup**
- **Purity constraints to be specified, where necessary, for packaged functions that are called from SQL**
- **Calling Java methods or C procedures from database triggers**
- **Location flexibility**

The Call Specification

The current way to publish external procedures is through call specifications. The call specification enables you to call external routines from other languages. Although the specification is designed for intercommunication between SQL, PL/SQL, C, and Java, it is accessible from any base language that can call these languages.

To use an already existing program as an external procedure, load, publish, and then call it.

Call specifications can be specified in any of the following locations:

- Stand-alone PL/SQL procedures and functions
- PL/SQL package specifications
- PL/SQL package bodies
- Object type specifications
- Object type bodies

Note: For functions that already have the pragma `RESTRICT_REFERENCES`, use the `TRUST` option. The SQL engine cannot analyze those programs to determine if they are free from side effects. The `TRUST` option makes it easier to call the Java and C procedures.

The Call Specification

- Identify the external body within a PL/SQL program to publish the external C procedure.

```
CREATE OR REPLACE FUNCTION function_name
(parameter_list)
RETURN datatype
  regularbody | externalbody
END;
```

- The external body contains the external C procedure information.

```
IS|AS LANGUAGE C
LIBRARY libname
[NAME C_function_name]
[CALLING STANDARD C | PASCAL]
[WITH CONTEXT]
[PARAMETERS (param_1, [param_n]);
```

Publishing an External C Routine

You create the PL/SQL procedure or function and use the `IS|AS LANGUAGE C` to publish the external C procedure. The external body contains the external routine information.

Syntax Definitions

where:	`LANGUAGE`	The language in which the external routine was written (defaults to C)
	`LIBRARY` *`libname`*	Name of the library database object
	`NAME` *`"C_function_name"`*	Represents the name of the C function; if omitted, the external procedure name must match the name of the PL/SQL subprogram
	`CALLING STANDARD`	Specifies the Windows NT calling standard (C or Pascal) under which the external routine was compiled (defaults to C)
	`WITH CONTEXT`	Specifies that a context pointer will be passed to the external routine for
	`parameters`	How arguments are passed to the external routine

The Call Specification

- The parameter list:

```
parameter_list_element
[ , parameter_list_element ]
```

- The parameter list element:

```
{ formal_parameter_name [indicator]
| RETURN INDICATOR
| CONTEXT }
[BY REFERENCE]
[external_datatype]
```

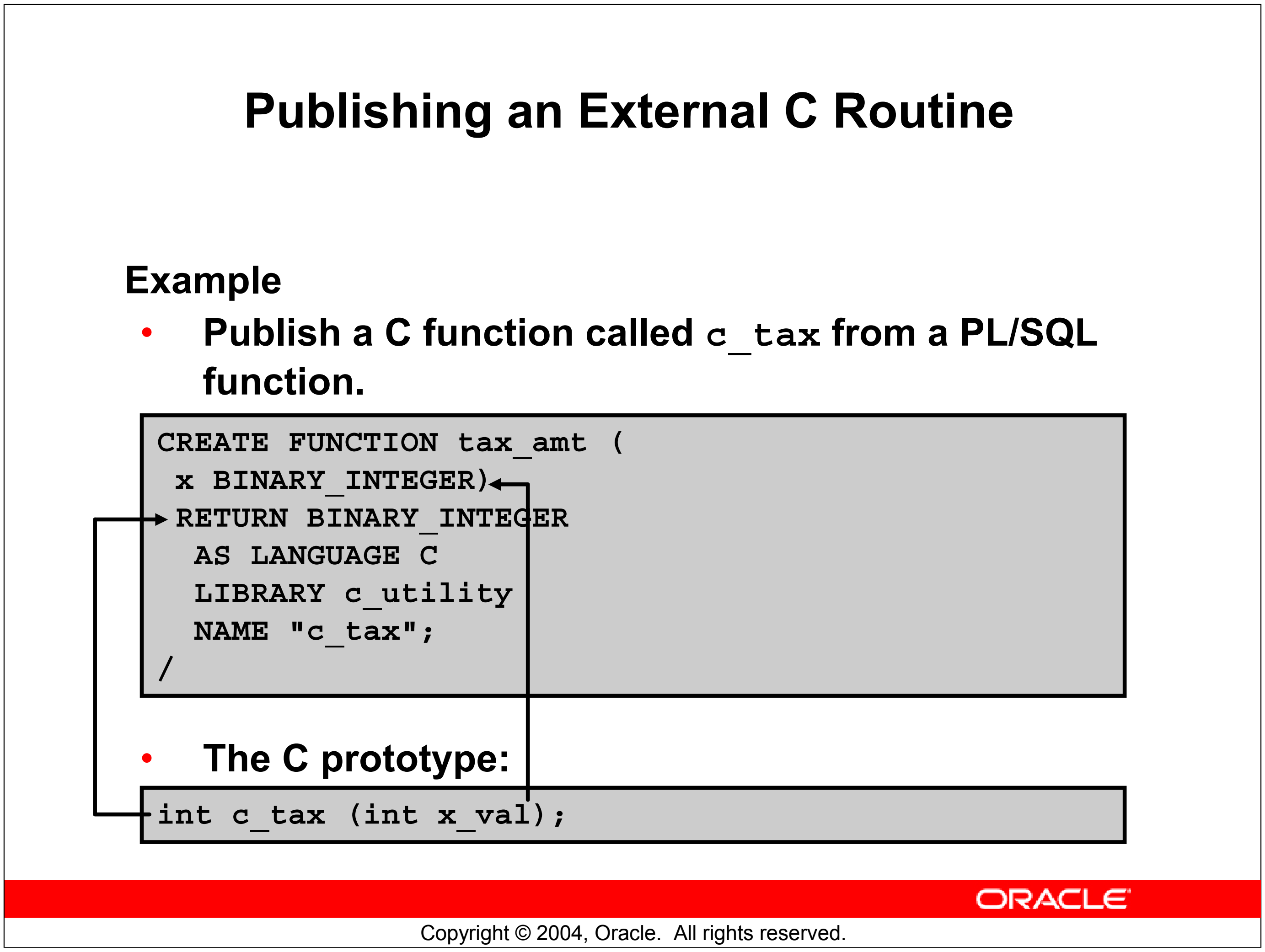
The `PARAMETER` Clause

The foreign parameter list can be used to specify the position and the types of arguments, as well as indicating whether they should be passed by value or by reference.

Syntax Definitions

where:	`formal_parameter_name [INDICATOR]`	Name of the PL/SQL parameter that is being passed to the external routine; the `INDICATOR` keyword is used to map a C parameter whose value indicates whether the PL/SQL parameter is null
	`RETURN INDICATOR`	Corresponds to the C parameter that returns a null indicator for the function
	`CONTEXT`	Specifies that a context pointer will be passed to the external routine
	`BY REFERENCE`	In C, you can pass `IN` scalar parameters by value (the value is passed) or by reference (a pointer to the value is passed). Use `BY REFERENCE` to pass the parameter by reference.
	`External_datatype`	The external data type that maps to a C data type

Note: The `PARAMETER` clause is optional if the mapping of the parameters is done on a positional basis, and indicators, reference, and context are not needed.



Publishing an External C Routine

Example

- **Publish a C function called `c_tax` from a PL/SQL function.**

```
CREATE FUNCTION tax_amt (
 x BINARY_INTEGER)
 RETURN BINARY_INTEGER
  AS LANGUAGE C
  LIBRARY c_utility
  NAME "c_tax";
/
```

- **The C prototype:**

```
int c_tax (int x_val);
```

Example

You have an external C function called `c_tax` that takes in one argument, the total sales amount. The function returns the tax amount calculated at 8%. The prototype for your `c_tax` function follows:

```
int c_tax (int x_val);
```

To publish the `c_tax` function in a stored PL/SQL function, use the `AS LANGUAGE C` clause within the function definition. The `NAME` identifies the name of the C function. Double quotation marks are used to preserve the case of the function defined in the C program. The `LIBRARY` identifies the library object that locates where the C file is. The `PARAMETERS` clause is not needed in this example because the mapping of the parameters is done on a positional basis.

Executing the External Procedure

1. Create and compile the external procedure in 3GL.
2. Link the external procedure with the shared library at the operating system level.
3. Create an alias library schema object to map to the operating system's shared library.
4. Grant execute privileges on the library.
5. Publish the external C procedure by creating the PL/SQL subprogram unit specification, which references the alias library.
6. Execute the PL/SQL subprogram that invokes the external procedure.

ORACLE

Executing the External Procedure: Example

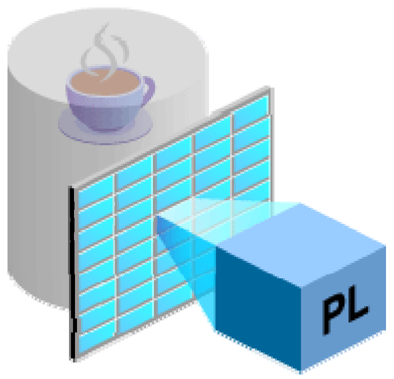
Call the external C procedure within a PL/SQL block:

```
DECLARE
  CURSOR cur_orders IS
    SELECT order_id, order_total
    FROM   orders;
  v_tax  NUMBER(8,2);
BEGIN
  FOR order_record IN cur_orders
  LOOP
    v_tax := tax_amt(order_record.order_total);
    DBMS_OUTPUT.PUT_LINE('Total tax: ' || v_tax);
  END LOOP;
END;
```

Overview of Java

The Oracle database can store Java classes and Java source, which:

- Are stored in the database as procedures, functions, or triggers
- Run inside the database
- Manipulate data

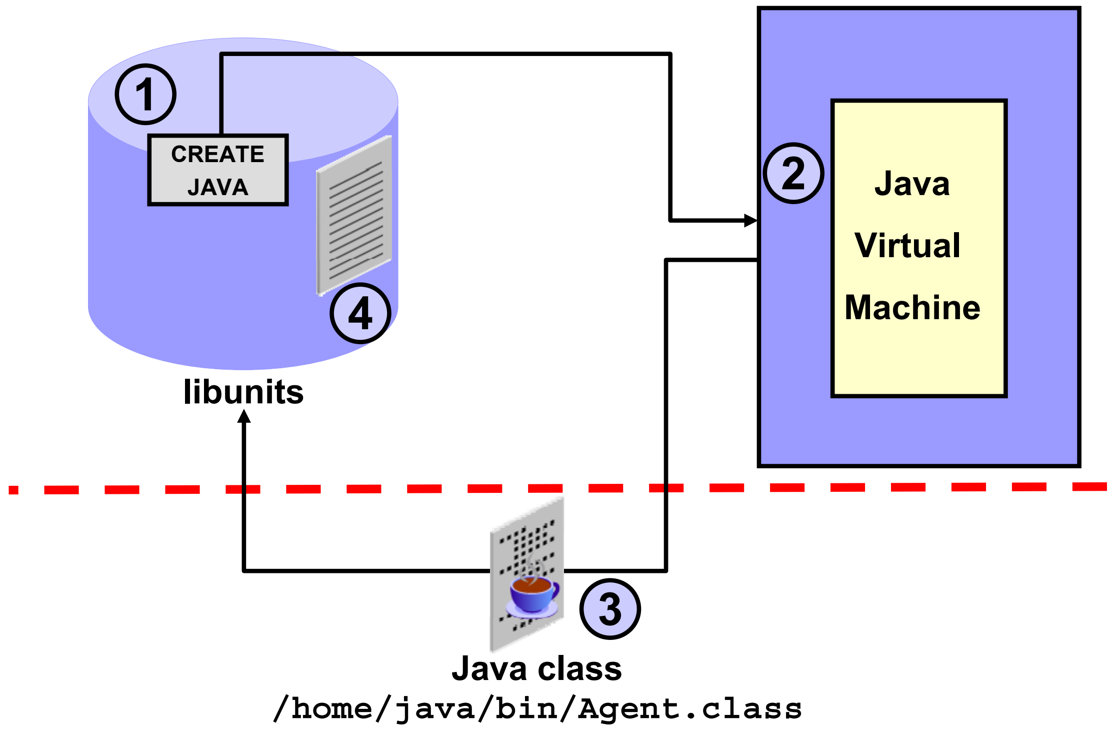


Java Overview

The Oracle database can store Java classes (`.class` files) and Java source code (`.java` files) and execute them inside the database, as stored procedures and triggers. These classes can manipulate data, but cannot display GUI elements such as AWT or Swing components. Running Java inside the database helps these Java classes to be called many times and manipulate large amounts of data, without the processing and network overhead that comes with running on the client machine.

You must write these named blocks and then define them by using the `loadjava` command or the SQL `CREATE FUNCTION`, `CREATE PROCEDURE`, `CREATE TRIGGER`, or `CREATE PACKAGE` statements.

How PL/SQL Calls a Java Class Method



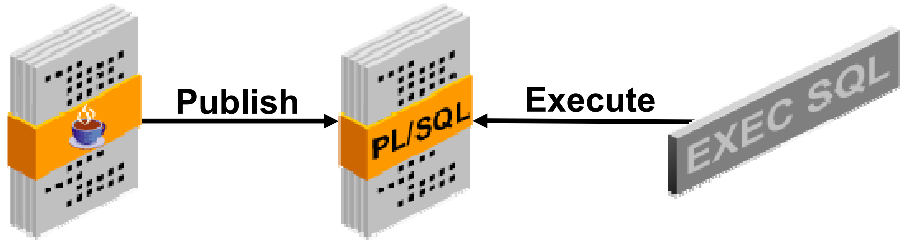
Calling a Java Class Method Using PL/SQL

The command-line utility `loadjava` uploads Java binaries and resources into a system-generated database table. It then uses the `CREATE JAVA` statement to load the Java files into RDBMS libunits. You can upload Java files from file systems, Java IDEs, intranets, or the Internet.

When the `CREATE JAVA` statement is invoked, the Java Virtual Machine library manager on the server loads Java binaries and resources from local `BFILE`s or `LOB` columns into RDBMS libunits. Libunits can be considered analogous to DLLs written in C, although they map one-to-one with Java classes, whereas DLLs can contain more than one routine.

Development Steps for Java Class Methods

1. **Upload the Java file.**
2. **Publish the Java class method by creating the PL/SQL subprogram unit specification that references the Java class methods.**
3. **Execute the PL/SQL subprogram that invokes the Java class method.**



ORACLE

Steps for Using Java Class Methods

Similar to using external C routines, the following steps are required to complete the setup before executing the Java class method from PL/SQL.

1. Upload the Java file. This takes an external Java binary file and stores the Java code in the database.
2. Publish the Java class method by creating the PL/SQL subprogram unit specification that references the Java class methods.
3. Execute the PL/SQL subprogram that invokes the Java class method.

Loading Java Class Methods

1. **Upload the Java file.**
   – **At the operating system, use the `loadjava` command-line utility to load either the Java class file or the Java source file.**

- **To load the Java class file, use:**

```
>loadjava -user oe/oe Factorial.class
```

- **To load the Java source file, use:**

```
>loadjava -user oe/oe Factorial.java
```

   – **If you load the Java source file, you do not need to load the Java class file.**

Loading Java Class Methods

Java classes and their methods are stored in RDBMS libunits in which you can load Java sources, binaries, and resources.

Use the `loadjava` command-line utility to load and resolve the Java classes. Using the `loadjava` utility, you can upload a Java source, class, and resource files into an Oracle database, where they are stored as Java schema objects. You can run `loadjava` from the command line or from an application.

After the file is loaded, it is visible in the data dictionary views.

```
SELECT object_name, object_type FROM   user_objects
WHERE  object_type like 'J%';
OBJECT_NAME                    OBJECT_TYPE
------------------------------ ------------------------
Factorial                      JAVA CLASS
SELECT text FROM   user_source WHERE name = 'Factorial';
TEXT
-------------------------------------------------------
public class Factorial {
  public static int calcFactorial (int n) {
    if (n == 1) return 1;
    else return n * calcFactorial (n - 1) ;
  }}
```

Publishing a Java Class Method

2. **Publish the Java class method by creating the PL/SQL subprogram unit specification that references the Java class methods.**
   - **Identify the external body within a PL/SQL program to publish the Java class method.**
   - **The external body contains the name of the Java class method.**

```
CREATE OR REPLACE
{  PROCEDURE procedure_name [(parameter_list)]
 | FUNCTION function_name [(parameter_list]...)]
   RETURN datatype}
   regularbody | externalbody
END;
```

```
{IS | AS} LANGUAGE JAVA
  NAME 'method_fullname (java_type_fullname
      [, java_type_fullname]...)
      [return java_type_fullname]';
```

Publishing a Java Class Method

The publishing of Java class methods is specified in the `AS LANGUAGE` clause. This call specification identifies the appropriate Java target routine, data type conversions, parameter mode mappings, and purity constraints. You can publish value-returning Java methods as functions and void Java methods as procedures.

Publishing a Java Class Method

- **Example:**

```
CREATE OR REPLACE FUNCTION plstojavafac_fun
  (N NUMBER)
  RETURN NUMBER
  AS
    LANGUAGE JAVA
    NAME 'Factorial.calcFactorial
       (int) return int';
```

- **Java method definition:**

```
public class Factorial {
  public static int calcFactorial (int n) {
    if (n == 1) return 1;
    else return n * calcFactorial (n - 1) ;
  }
}
```

Example

If you want to publish a Java method named `calcFactorial` that returns the factorial of its argument, as explained in the preceding example:

- The `NAME` clause string uniquely identifies the Java method
- The PL/SQL function shown corresponds with regard to the parameters
- The parameter named `N` corresponds to the `int` argument

Executing the Java Routine

1. Upload the Java file.
2. Publish the Java class method by creating the PL/SQL subprogram unit specification that references the Java class methods.
3. Execute the PL/SQL subprogram that invokes the Java class method.

Example (continued)

You can call the `calcFactorial` class method by using the following command:

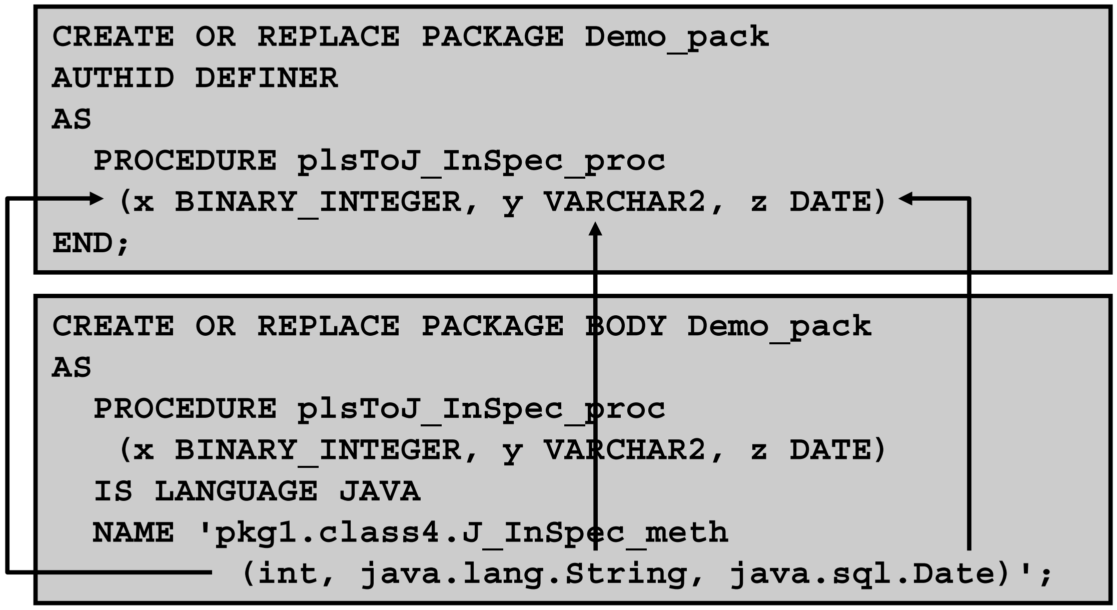
```
EXECUTE DBMS_OUTPUT.PUT_LINE(plstojavafac_fun (5))
120
```

Alternatively, to execute a `SELECT` statement from the `DUAL` table:

```
SELECT plstojavafac_fun (5)
FROM dual;

PLSTOJAVAFAC_FUN(5)
-------------------
                120
```

Creating Packages for Java Class Methods



```
CREATE OR REPLACE PACKAGE Demo_pack
AUTHID DEFINER
AS
  PROCEDURE plsToJ_InSpec_proc
   (x BINARY_INTEGER, y VARCHAR2, z DATE)
END;
```

```
CREATE OR REPLACE PACKAGE BODY Demo_pack
AS
  PROCEDURE plsToJ_InSpec_proc
   (x BINARY_INTEGER, y VARCHAR2, z DATE)
  IS LANGUAGE JAVA
  NAME 'pkg1.class4.J_InSpec_meth
          (int, java.lang.String, java.sql.Date)';
```

Creating Packages for Java Class Methods

These examples create a package specification and body `Demo_pack`.

The package is a container structure. It defines the specification of the PL/SQL procedure named `plsToJ_InSpec_proc`.

Note that you cannot tell whether this procedure is implemented by PL/SQL or by way of an external procedure. The details of the implementation appear only in the package body in the declaration of the procedure body.

Summary

In this lesson, you should have learned how to:

- Use external C routines and call them from your PL/SQL programs
- Use Java methods and call them from your PL/SQL programs

Summary

You can embed calls to external C programs from your PL/SQL programs by publishing the external routines in a PL/SQL block. You can take external Java programs and store them in the database to be called from PL/SQL functions, procedures, and triggers.

Practice Overview

This practice covers the following topics:

- Writing programs to interact with C routines
- Writing programs to interact with Java code

Practice Overview

In this practice, you will write two PL/SQL programs. One program calls an external C routine and the second program calls a Java routine.

Practice 4

1. An external C routine definition is created for you. The `.c` file is stored in the `$HOME/labs` directory on the database server. This function returns the tax amount based on the total sales figure passed to the function as a parameter. The name of the `.c` file is named `calc_tax.c`. The shared object file name is `calc_tax.so`. The function is defined as:

```
calc_tax(n)
int n;
{
 int tax;
 tax=(n*8)/100;
 return(tax);
}
```

   a. Create `calc_tax.so` file by using the following command:

```
cc -shared -o calc_tax.so calc_tax.c
```

   b. Copy the file `calc_tax.so` to `$ORACLE_HOME/bin` directory using the following command:

```
cp calc_tax.so $ORACLE_HOME/bin
```

   c. Create the library object. Name the library object `c_code` and define its path as:

```
CREATE OR REPLACE LIBRARY c_code
 AS '$ORACLE_HOME/bin/calc_tax.so';
/
```

   d. Publish the external C routine.
   Create a function named `call_c`. This function has one numeric parameter and it returns a binary integer. Identify the `AS LANGUAGE`, `LIBRARY`, and `NAME` clauses of the function.

   e. Create a procedure to call the `call_c` function created in the previous step.
   Name this procedure `C_OUTPUT`. It has one numeric parameter. Include a `DBMS_OUTPUT.PUT_LINE` statement so that you can view the results returned from your C function.

   f. Set serveroutput ON and execute the `C_OUTPUT` procedure.

2. A Java method definition is created for you. The method accepts a 16-digit credit card number as the argument and returns the formatted credit card number (4 digits followed by a space). The name of the `.class` file is `FormatCreditCardNo.class`. The method is defined as:

```
public class FormatCreditCardNo
{
public static final void formatCard(String[] cardno)
{
int count=0, space=0;
String oldcc=cardno[0];
String[] newcc= {""};
while (count<16)
{
newcc[0]+= oldcc.charAt(count);
space++;
if (space ==4)
{  newcc[0]+=" "; space=0;  }
count++;
}
cardno[0]=newcc [0];
}
}
```

a. Load the `.java` source file.

b. Publish the Java class method by defining a PL/SQL procedure named `CCFORMAT`. This procedure accepts one `IN OUT` parameter.

   Use the following definition for the `NAME` parameter:
   ```
   NAME 'FormatCreditCardNo.formatCard(java.lang.String[])';
   ```

c. Execute the Java class method. Define one SQL*Plus variable, initialize it, and use the `EXECUTE` command to execute the `CCFORMAT` procedure.

```
EXECUTE ccformat(:x);


PRINT x
                   X
-------------------
1234 5678 1234 5678
```