Managing PLSQL Code

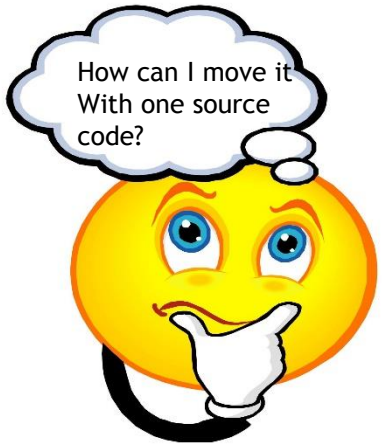
- Describe and use conditional compilation
- Hide PL/SQL source code using dynamic obfuscation and the Wrap utility

What Is Conditional Compilation?

This option started in 11g

Source code

```
create or replace function get_sal
(p_emp_id number)
return number Result_cashe
is
v_sal number;
begin
  select salary into v_sal
  from employees
  where employee_id=p_emp_id;

  return v_sal;

end;
```



Customer use 10g

Customer use 11g

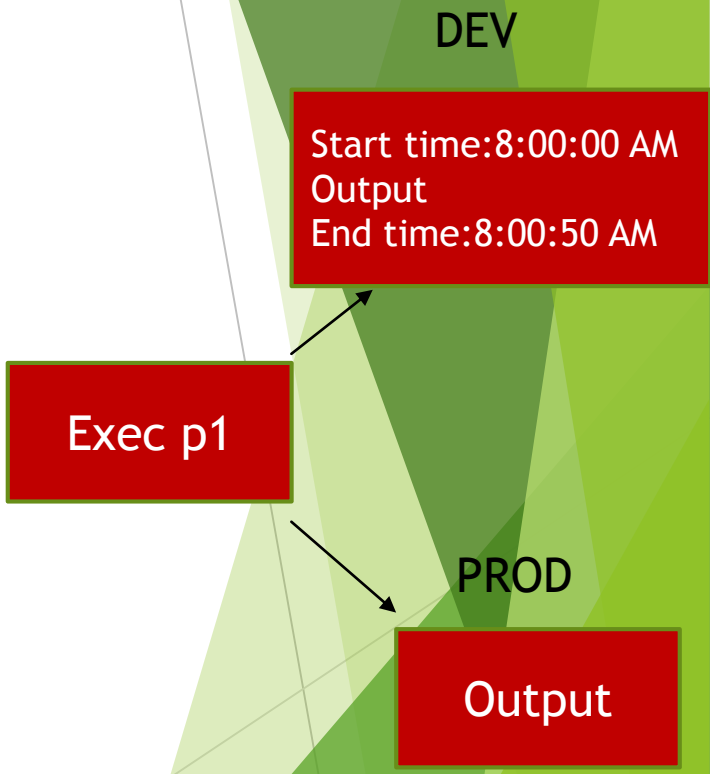
What Is Conditional Compilation?

Enables you to customize the functionality in a PL/SQL application without removing any source code:

- Utilize the latest functionality with the latest database release or disable the new features to run the application against an older release of the database.
- Activate debugging or tracing functionality in the development environment and hide that functionality in the application while it runs at a production site.

```
$IF, $THEN, $ELSE,
$ELSIF, $END, $$, $ERROR
```

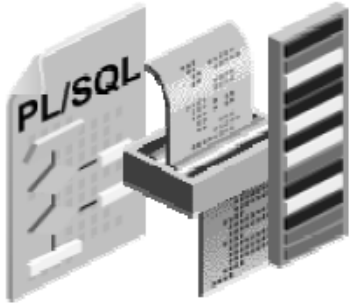
Reserved preprocessor control tokens



Benefits of Conditional Completion

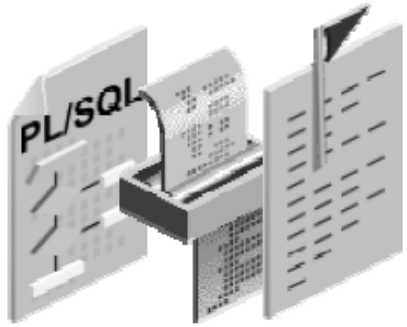
- Support for multiple versions of the same program in one source code
- Easy maintenance and debugging of code.
- Easy Migration of code to a different release of the database.

How Does Conditional Compilation Work?



Selection directives:
Use the `$IF` token.

Inquiry directives:
Use the `$$` token.



Error directives:
Use the `$ERROR` token.

`DBMS_PREPROCESSOR`
package

`DBMS_DB_VERSION`
package

The DBMS_DB_VERSION Package

Introduce in Oracle10g release 2

```
SELECT text
from ALL_source
WHERE lower(name)='dbms_db_version'
order by line
```

```
package dbms_db_version is

  version constant pls_integer := 12;

  release constant pls_integer := 1;

  ver_le_9_1     constant boolean := FALSE;

  ver_le_9_2     constant boolean := FALSE;

  ver_le_9       constant boolean := FALSE;

  ver_le_10_1    constant boolean := FALSE;

  ver_le_10_2    constant boolean := FALSE;

  ver_le_10      constant boolean := FALSE;

  ver_le_11_1    constant boolean := FALSE;

  ver_le_11_2    constant boolean := FALSE;

  ver_le_11      constant boolean := FALSE;

  ver_le_12_1    constant boolean := TRUE;

  ver_le_12      constant boolean := TRUE;

end dbms_db_version;
```

```
begin
dbms_output.put_line(dbms_db_version.version);
dbms_output.put_line(dbms_db_version.release);
  if   dbms_db_version.ver_le_12  or   dbms_db_version.ver_le_12_1 then
  dbms_output.put_line('it is 12c :)');
  end if;

end;
```

Using Selection Directives

```
$IF <Boolean-expression> $THEN Text
$ELSEIF <Boolean-expression> $THEN Text
. . .
$ELSE Text
$END
```

```
create or replace function get_sal
(p_emp_id number)
return number  $if dbms_db_version.version>=11 $then result_cache $end
is
v_sal number;
begin
  select salary into v_sal
  from employees
  where employee_id=p_emp_id;

  return v_sal;

end;
```

Using Predefined and User-Defined Inquiry Directives

Predefined inquiry directives

```
PLSQL_CCFLAGS
PLSQL_CODE_TYPE
PLSQL_DEBUG
PLSQL_OPTIMIZE_LEVEL
PLSQL_WARNINGS
NLS_LENGTH_SEMANTICS
PLSQL_LINE
PLSQL_UNIT
```

First it will search this value here

Then here

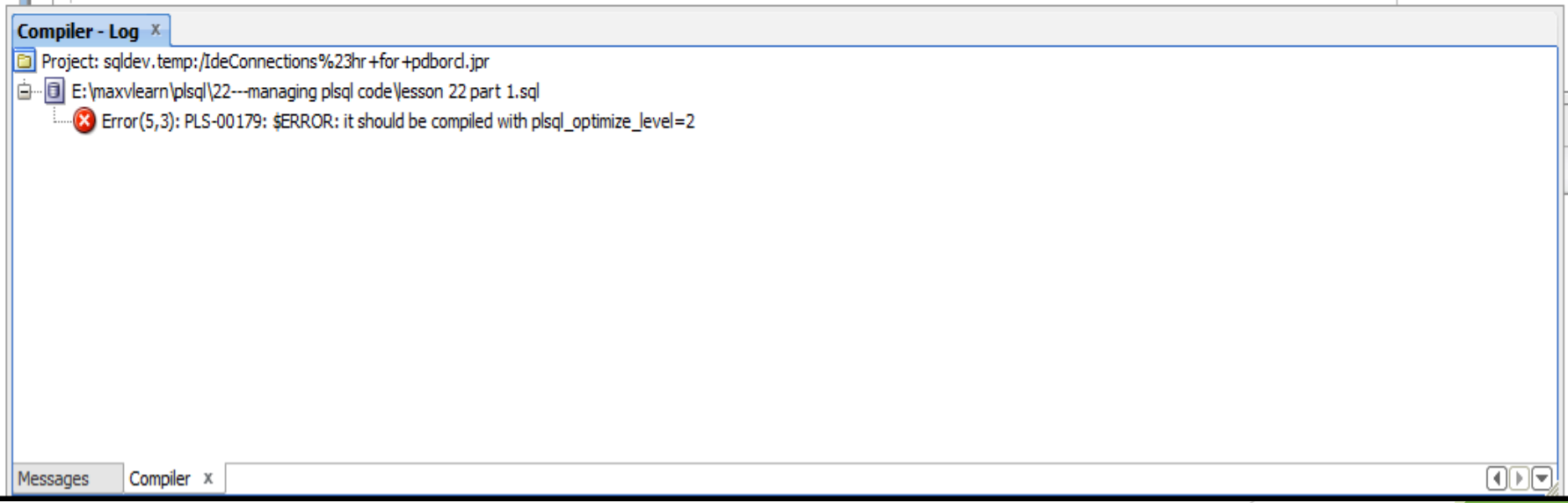
```
Inquiry Directives

DBMS_OUTPUT.PUT_LINE($$PLSQL_LINE);


Error Directives
$IF $$PLSQL_OPTIMIZE_LEVEL != 2
$THEN
   $ERROR 'intensive_program must be compiled with
   maximum optimisation'
 $END
$END
```

User-defined inquiry directives

```
PLSQL_CCFLAGS = 'plsql_ccflags:true,debug:true,debug:0';
```

Example

```
create or replace procedure g_test
is
begin
  $if $$plsql_optimize_level <>2 $then
  $error 'it should be compiled with plsql_optimize_level=2 ' $end
  $end
dbms_output.put_line('test');
end;
```



When you search for any $$ value ????
example : $$debug

```
PLSQL_CCFLAGS
PLSQL_CODE_TYPE
PLSQL_DEBUG
PLSQL_OPTIMIZE_LEVEL
PLSQL_WARNINGS
NLS_LENGTH_SEMANTICS
PLSQL_LINE
PLSQL_UNIT
```

The following describes the order of the processing flow when conditional compilation attempts to resolve an inquiry directive:

1. The *id* is used as an inquiry directive in the form $$*id* for the search key.

2. The two-pass algorithm proceeds as follows:

   The string in the PLSQL_CCFLAGS initialization parameter is scanned from right to left, searching with *id* for a matching name (case insensitive); done if found.

   The predefined inquiry directives are searched; done if found.

3. If the $$*id* cannot be resolved to a value, then the PLW-6003 warning message is reported if the source text is not wrapped. The literal NULL is substituted as the value for undefined inquiry directives. Note that if the PL/SQL code is wrapped, then the warning message is disabled so that the undefined inquiry directive is not revealed.

The PLSQL_CCFLAGS Parameter and the Inquiry Directive

Use the PLSQL_CCFLAGS parameter to control conditional compilation of each PL/SQL library unit independently.

```
PLSQL_CCFLAGS = '<v1>:<c1>,<v2>:<c2>,...,<vn>:<cn>'
```

```
ALTER SESSION SET
PLSQL_CCFLAGS = 'plsql_ccflags:true, debug:true, debug:0';
```

- <vi> has the form of an unquoted PL/SQL identifier. It is unrestricted and can be a reserved word or a keyword. The text is insensitive to case. Each one is known as a flag or flag name. Each <vi> can occur more than once in the string, each occurrence can have a different flag value, and the flag values can be of different kinds.
- <ci> is one of the following: a PL/SQL boolean literal, a PLS_INTEGER literal, or the literal NULL. The text is insensitive to case. Each one is known as a flag value and corresponds to a flag name.

Displaying the PLSQL_CCFLAGS Initialization Parameter Setting

```
SELECT name, type, plsql_ccflags
FROM   user_plsql_object_settings
```

Results:

	NAME	TYPE	PLSQL_CCFLAGS
1	DEPT_PKG	PACKAGE	(null)
2	DEPT_PKG	PACKAGE BODY	(null)
3	TAXES_PKG	PACKAGE	(null)
4	TAXES_PKG	PACKAGE BODY	(null)
5	EMP_PKG	PACKAGE	(null)
6	EMP_PKG	PACKAGE BODY	(null)
7	SECURE_DML	PROCEDURE	(null)
8	SECURE_EMPLOYEES	TRIGGER	(null)
9	ADD_JOB_HISTORY	PROCEDURE	plsql_ccflags:true, debug:true, debug:0
10	UPDATE_JOB_HISTORY	TRIGGER	(null)

Using DBMS_PREPROCESSOR Procedures to Print or Retrieve Source Text

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (
   object_type     VARCHAR2,
   schema_name     VARCHAR2,
   object_name     VARCHAR2);
```

lesson 22 part 3.sql

Worksheet | Query Builder

```
create or replace procedure test_only
is
v varchar2($$maxsizev);
begin
    if $$trace_time then
    dbms_output.put_line(to_char(sysdate,'hh:mi:ss') );
    end if;

    for i in 1..1000
    loop
    dbms_output.put_line($$maxsizev);
    end loop;


    if $$trace_time then
    dbms_output.put_line(to_char(sysdate,'hh:mi:ss') );
    end if;

end;
```

1

```
call dbms_preprocessor.print_post_processed_source('procedure', 'hr','test_only');
```

2

Dbms Output

Buffer Size: 20000

```
procedure test_only
is
v varchar2( 2000     );
begin
    if  FALSE       then
    dbms_output.put_line(to_char(sysdate,'hh:mi:ss') );
    end if;

    for i in 1..1000
    loop
    dbms_output.put_line( 2000     );
    end loop;


    if  FALSE       then
    dbms_output.put_line(to_char(sysdate,'hh:mi:ss') );
    end if;

end;
```

3

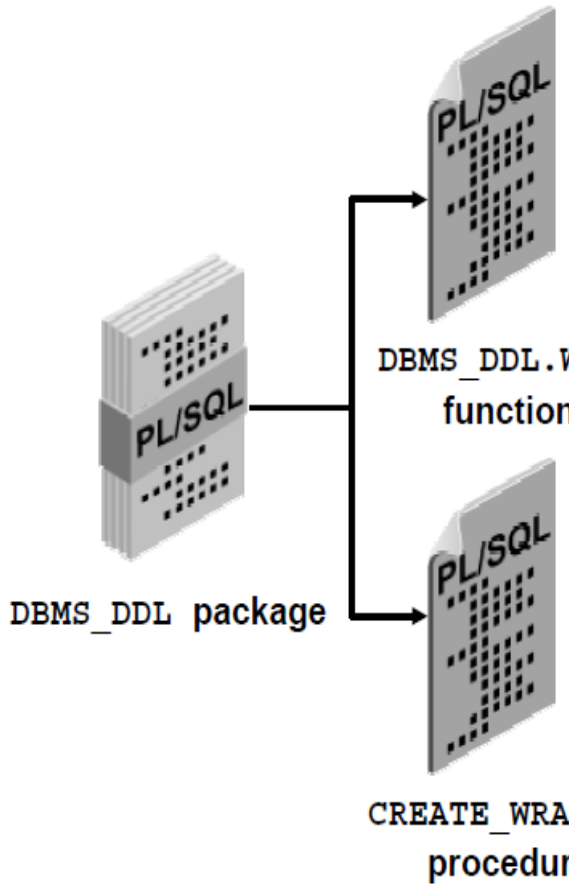
What Is Obfuscation?

- Obfuscation (or wrapping) of a PL/SQL unit is the process of hiding the PL/SQL source code.
- Wrapping can be done with the wrap utility and `DBMS_DDL` subprograms.
- The wrap utility is run from the command line and it processes an input SQL file, such as a SQL*Plus installation script.
- The `DBMS_DDL` subprograms wrap a single PL/SQL unit, such as a single `CREATE PROCEDURE` command, that has been generated dynamically.

Benefits of Obfuscating

- It prevents others from seeing your source code.
- Your source code is not visible through the `USER_SOURCE`, `ALL_SOURCE`, or `DBA_SOURCE` data dictionary views.
- SQL*Plus can process the obfuscated source files.
- The `Import` and `Export` utilities accept wrapped files.

What's New in Dynamic Obfuscating Since Oracle 10*g*?



Provides the same functionality as the `CREATE_WRAPPED` procedure but allows for larger inputs

Wraps the text and creates the PL/SQL unit

dbms_ddl.create_wrapped

```
begin
dbms_ddl.create_wrapped
(
       'create or replace function get_sum_sal_dept
       ( dept_id number )
       return number
       is
       v_sal number;
       begin
         select sum(salary)
         into v_sal
         from
         employees
         where department_id =dept_id;
         return v_sal;

        exception
        when others then return null;
       end;'
);
end;
```

The text for the function Between single Quotation

```
select line,text
from user_source
where lower(name)='get_sum_sal_dept';
```

```
function get_sum_sal_dept wrapped
a000000
b2
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
8
145 10b
1gOOi+i/dGdOZYThVtI3Lh8mbjEwg+nI18usfC9GAE6OEkYXyhzOk6ChmGxzzK0qG9OW7HWp
u5aPRTMlKcTr+qBMGhsWDLgswSHtT95FIiV5z9eIftUHb/jIw5mOLvDLnqYcrXSz88fJlwNX
bA/eQY1HJ3oW4LKPzAu23YC3ljyXsgJfjkT4Li93rfZdqeixn2fS5v+A9VMrO3F8WrruAHiW
lZXwxJFrnaSW1wYPnWb4B9O0S6IS5wqnH/a+SnenmCg2X1qG
```

The PL/SQL Wrapper Utility

- The PL/SQL wrapper is a stand-alone utility that hides application internals by converting PL/SQL source code into portable object code.
- Wrapping has the following features:
  - Platform independence
  - Dynamic loading
  - Dynamic binding
  - Dependency checking
  - Normal importing and exporting when invoked

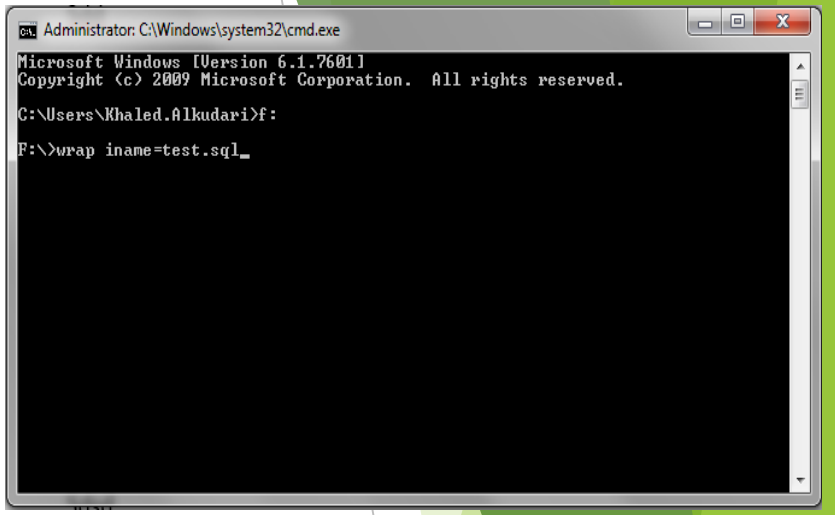
Running the Wrapper Utility

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

- Do not use spaces around the equal signs.
- The `INAME` argument is required.
- The default extension for the input file is `.sql`, unless it is specified with the name.
- The `ONAME` argument is optional.
- The default extension for output file is `.plb`, unless specified with the `ONAME` argument.

```
WRAP INAME=demo_04_hello.sql
WRAP INAME=demo_04_hello
WRAP INAME=demo_04_hello.sql ONAME=demo_04_hello.plb
```

cmd



Wrap work only for:
Type
Type body
Procedure
Function
Package
Package body

Guidelines for Wrapping

- You must wrap only the package body, not the package specification.
- The wrapper can detect syntactic errors but cannot detect semantic errors.
- The output file should not be edited. You maintain the original source code and wrap again as required.
- To ensure that all the important parts of your source code are obfuscated, view the wrapped file in a text editor before distributing it.

The DBMS_DDL Package Versus the Wrap Utility

Functionality	DBMS_DLL	Wrap Utility
Code obfuscation	Yes	Yes
Dynamic Obfuscation	Yes	No
Obfuscate multiple programs at a time	No	Yes

Thank You