“This presentation is for informational purposes only and may not be incorporated into a contract or agreement.”
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decision. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
THE INFORMATION COMPANY
Bryn Llewellyn

PL/SQL Product Manager, Oracle Headquarters
What’s new in PL/SQL?

Hear it from the Product Manager.

And ask its Developers.
Agenda

• Preamble

• The real talk
Preamble #1 – Ask Oracle’s PL/SQL Team

- They’re here in the room
- We’ll have a Q&A here at the end
- We’ll walk over the road to the OTN lounge
- We can continue there for as long as you want
Preamble #2 – Can’t get enough of PL/SQL?

- A two-day conference packed with intensive training on the PL/SQL language
- Six weeks from today
- I’m doing two talks

- Sponsored by O'Reilly Media, Quest Software, and ODTUG

More information available at www.oracleplsqlprogramming.com
And there’s fliers here in this room
Preamble #3 – Feel free to use these slides

- *Do* use them to teach your colleagues
- *Don’t* use them as the primary source of the material – they have no notes
- Instead, use my accompanying whitepaper
- And study the account in the Oracle Database Documentation Library
Preamble #4 –
New PL/SQL Features in 10.2

...besides the one that’s
the main subject of this talk!
Utl_Nla

- Provides a PL/SQL API to a linked-in C implementation of two of the most popular available libraries for matrix math: BLAS and LAPACK

```sql
select distinct Object_Name from All_Arguments
    where Package_Name = 'UTL_NLA'
    and Owner = 'SYS'
    and (Object_Name like '%BLAS%' or
    Object_Name like '%LAPACK%')
```

- 33 BLAS overloaded subprograms
- 23 LAPACK overloaded subprograms
Dynamic obfuscation

• The problem

```sql
begin
    execute immediate q'
    
    create or replace procedure P is
        begin
            DBMS_Output.Put_Line (q'[I'm not wrapped]');
        end P;
    end;

end;
```

• The All_Source view family shows the source in plain text
DBMS_DDL.Create_Wrapped

- The solution

```plsql
begin
    DBMS_DDL.Create_Wrapped ( q'
            create or replace procedure P is
            begin
                DBMS_Output.Put_Line (q'[I'm wrapped now]');
                end P;
            }' );
end;
```

- The `All_Source` view family now obfuscates the source
DBMS_DDL.Create_Wrapped

```sql
select Text from User_Source
    where Name = 'P' order by Line
```

```plaintext
procedure P wrapped
    a000000
    2
    abcd
    abcd
    ...
    abcd
    abcd
    7
    5a 92
cxX4swY3RKBC0enZsNOP1N09CC4wg5nnm7+fMr2ywFznaaV0iwmm4Unqv64kfAw1r5X6eFcZ
JCEUyiGiKOOGEHpzczQhnj2MAD3H1Jdt05meCV60m5vZCz0qO6+jcXNjuFnWTmmnqZvyA==
```
DBMS_Output

SELECT Text FROM All_Source
WHERE Owner = 'SYS'
AND Name = 'DBMS_OUTPUT'
AND Lower(Text) LIKE '%type%chararr%'

- In 10.1

TYPE CHARARR IS
  TABLE OF VARCHAR2(255) INDEX BY BINARY_INTEGER;

- In 10.2

TYPE CHARARR IS
  TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
CONNECT Usr/p@Rel_10_2

-- Needs the 10.2 SQL*Plus (of course)
SET SERVEROUTPUT ON SIZE UNLIMITED

• This is the default, of course
The real talk…

PL/SQL Conditional Compilation
Conditional Compilation

- What’s the elevator pitch?
- What’s it good for?
- What does it look like?
- What’s the terminology?
- Use cases & best practices
The elevator pitch

• It’s part of the syntax and semantics of the PL/SQL language
• Looks very similar to the regular if construct
• Yet it’s dramatically different in its meaning
• It supports many new exciting solutions to historical programming challenges
• It allows new best practices to be defined
The elevator pitch

• The regular if selects action at run-time
• Conditional compilation selects text at compile-time
• Unselected text needn’t be legal
• It can select declarations
• It can interrupt regular statements
What’s it good for?

- It lets self-tracing code – and assertions – be turned on during development and turned off when the code goes live
- It lets developers prototype alternative implementations with increased productivity and reduced risk of error
- It enables new approaches to unit testing
- It gives ISVs new mechanisms for component-based installation
- It lets ISVs use a single code corpus in many Oracle Database releases
“We found PL/SQL conditional compilation functionally complete and easy to use.

It will allow us to write a more manageable and faster implementation for our component-based architecture.

We intend to use it in our products at the earliest opportunity.”

— Håkan Arpfors, Senior Software Architect, IFS,

www.ifsworld.com
“I described the challenge we faced with unit testing our package-body-private subprograms to Bryn before I knew that PL/SQL conditional compilation was on the way.

Testing is critical to us and I'm excited to see the new possibilities that this feature gives us — both for ordinary unit testing and for the PL/SQL equivalent of mock objects.”

— Nick Strange, Principal Architect, Fidelity Brokerage Company Technology, www.fidelity.com
if Control.Trace_Level > 0 then
    Print(Sparse_Collection.Count());
    if Control.Trace_Level > 1 then
        declare Idx Idx_t := Sparse_Collection.First();
        begin
            while Idx is not null loop
                Print(Idx||' '||Sparse_Collection(Idx));
                Idx := Sparse_Collection.Next(Idx);
            end loop;
        end;
        end if;
    end if;
end if;
What does it look like?

```$if$ Control.Trace_Level > 0 $then$
Print(Sparse_Collection.Count());
$if$ Control.Trace_Level > 1 $then$
    declare Idx Idx_t := Sparse_Collection.First();
    begin
        while Idx is not null loop
            Print(Idx||' '||Sparse_Collection(Idx));
            Idx := Sparse_Collection.Next(Idx);
        end loop;
    end;
$end$
$end$
Hold on…

...there’s more to it.
procedure P is
begin
   if Control.Tracing then
      Print('Tracing');
   end
end P;

package Control is
   Tracing : constant boolean := true;
end Control;
package Control is
    Tracing constant boolean := Sysdate < '22-Sep-05';
...
    Var     constant pls_integer := Pkg.F(42);
end Control;

procedure P is
begin
    $if Control.Tracing $then
        Print('Tracing');
    $end
end P;
No, that would be crazy!

PLS-00174: a **static boolean expression** must be used

- When a package constant controls what text is selected for compilation...
- ...the selection will never be changed
- ...unless the controlling package is recompiled by hand
- Notice that a dependency is set up
What’s the terminology?

• The conditional compilation construct that we’ve looked at...

• ...loosely speaking, the compile-time $if$ construct...

• ...is called the *selection directive*
Is a package the only way to control it?

• Hmm… It seems a bit heavy-handed to have to create a partner package for every compilation unit I want to conditionalize
• Relax… There’s a lightweight way
• Sometimes the lightweight way is better
• Sometimes the package way is better
• We’ll soon see when and why to use which
alter session set P_\text{Plsql\_CCFlags} = 'Tracing\text{\:true}'
/
create or replace procedure P is
begin
\text{\$if}\ $\text{\$Tracing}\ $\text{\$then}$
Print('Tracing');
\text{\$end}$
end P;
/

select \text{\text{\text{P_\text{Plsql\_CCFlags}} from User\_Plsql\_Object\_Settings}}
where Name = 'P'
/
What’s the terminology?

• **\$\$Tracing** is an example of an *inquiry directive*

• An *inquiry directive* gets a value from the compilation environment

• Think “command line”

• Not only user-defined *ccflags* like *Tracing*

• Also pre-defined *Plsql_Unit* and *Plsql_Line*

• Also the PL/SQL compilation parameters like *Plsql_Optimize_Level*
What about “case not found”? 

```
alter session set Plsql_CCCFlags = ' Trace_Level:3 ' /
create procedure P is
begin
    $if $$Trace_Level = 0 $then ...;
    $elsif $$Trace_Level = 1 $then ...;
    $elsif $$Trace_Level = 2 $then ...;
    $else $error 'Bad: '||$$Trace_Level $end
$end
end P;
/

PLS-00179: $ERROR: Bad: 3
```
What’s the terminology?

- `$error ... $end` is the `error directive`
- And that’s it:
  - `selection directive`
  - `inquiry directive`
  - `error directive`
  - `static package constant`
  - `static boolean expression`
  - `PL/SQL compilation parameter`
  - `ccflag`
How do I know what got compiled?

procedure P is
begin
    $if $$Tracing $then
        Print('Tracing');
    $else
        Print('Not tracing');
    $end
end P;
alter procedure P compile
    Plsql_CCFlags = 'Tracing:true'
begin
    DBMS_Preprocessor.Print_Post_Processed_Source(
        Schema_Name => 'USR',
        Object_Type => 'PROCEDURE',
        Object_Name => 'P');
end;
/
alter procedure P compile

  Plsql_CCFlags = 'Tracing:NULL' reuse settings
/
begin
  DBMS_Preprocessor.Print_Post_Processed_Source(
    Schema_Name => 'USR',
    Object_Type => 'PROCEDURE',
    Object_Name => 'P');
end;
/

procedure P is
begin
    Print('Tracing');
end P;
DBMS_Preprocessor shows...

```plaintext
procedure P is
begin
    Print('Not tracing');
end P;
```
What if a ccflag is not defined?

```
alter session set Plsql_CCFlags = ''
/
alter session set Plsql_Warnings = 'Enable:All'
/
create or replace procedure P is
begin
  $if $$Tracing $then
    Print('Tracing');
  $else
    Print('Not tracing');
  $end
end P;
/
SHOW ERRORS
begin P(); end;
/```
What if a `ccflag` is not defined?

- You get a warning
  
  `PLW-06003: unknown inquiry directive '$$$TRACING'`

- It evaluates to null
  
  `Not tracing`

- This is very useful – remember it for later
And now, on to the use cases...
Tracing and assertions

• For many, the *defining* use case for conditional compilation
  • Assertions can... be a form of documentation: they can describe the state the code expects to find before it runs (its preconditions), and the state the code expects to result in when it is finished running (postconditions). Assertions are also sometimes placed at points the execution is not supposed to reach.
  • The removal of assertions from production code is almost always done automatically. It usually is done via conditional compilation.

• [en.wikipedia.org/wiki/Assertion_(computing)](en.wikipedia.org/wiki/Assertion_(computing))
Tracing and assertions

- Tracing outputs information to help you get your program correct
- Assertions ensure – without output – that your program state is correct at critical points in the execution flow (else abort)
- Similar because you want each enabled at development time and disabled when the code goes live
- You use conditional compilation in the same way for both
Tracing needs its own support

for j in 1..Records.Last() loop
    $if $$Tracing $then
        Tracing_Counter := Tracing_Counter + 1;
        if Tracing_Counter > $$Tracing_Step then
            $if Ok(Records(j)) then
                Show_Record(Records(j));
                Show_Status(Ok(Records(j)));
                Tracing_Counter := 0;
            $end if;
        $end if;
    $end

    if Ok(Records(j)) then
        ...
    end if;
end loop;
procedure P is
  ...
  $if $$Tracing $then
    Tracing.Counter pls_integer := 0;
  $end
  ...
  $if $$Tracing $then
    procedure Show_Record(i in Rt) is
      begin ... end Show_Record;
    procedure Show_Status(i in boolean) is
      begin ... end Show_Status;
  $end
begin
Unit testing of body-private subprograms

• Many programmers believe that it’s not sufficient just to let body-private helpers be tested implicitly as a side-effect of testing the package’s public API.
• They have been forced to declare would-be private subprograms in the package spec.
• This causes problems.
Unit testing of body-private subprograms

- Conditional compilation gives you two new approaches to choose between

- You can conditionally expose the private subprograms for external testing

- You can write all the tests inside the package, guard them all with selection directives, and expose them all conditionally via a single `Run_The_Tests()` procedure
package body Pkg is
  procedure P1(...) is ... end P1;
  ...
  procedure Helper1(...) is ... end Helper1;
  ...
  procedure Helper1_(...) is begin
    $if $$Testing $then
      Helper1(...);
    $else
      raise Program_Error;
    $end
  end Helper1_;
  ...
end Pkg;
...via a conditionally usable wrapper in the spec

package Pkg is
    procedure P1(...);
    ...
    procedure Helper1_(...);
    ...
end;
Encapsulate the testing...

```plaintext
package body Pkg is
    procedure P1(...) is ... end P1;
    ...
    procedure Helper1(...) is ... end Helper1;
    ...
    procedure Run_The Tests(...) is
        begin
            $if $$Testing $then
                P1(...);
                Helper1(...);
                ...
            $else
                raise Program_Error;
            $end
        end Run_The Tests;
end Pkg;
```
...and expose it – conditionally usable – in the spec

package Pkg is
    procedure P1(...);
    ...
    procedure Run_The Tests(...);
end;
Encapsulating the testing

- Allows the testing code to set up body-private state
- Allows the unit testing of arbitrarily deeply nested inner subprograms, e.g. `Deep_P()`
- These aren’t visible in an outer scope
- Therefore, write its test at the same nesting level as `Deep_P()` — guarded, of course, by a selection directive
- Provide a conditionally guarded path to invoke the test from `Run_The_Tests()`
Mock objects

• Borrowing the term from object-oriented programming as a mnemonic for the current use case

• It denotes a paradigm for unit testing
  • In tests, a mock object behaves exactly like a real object with one crucial difference: the programmer will hard-code return values for its methods...

• en.wikipedia.org/wiki/Mock_Object
Mock objects

• Subprogram `To_Be_Tested()` calls `Callee()`
• `To_Be_Tested()` must behave correctly in response to $N$ distinct legal “patterns” returned by `Callee()` and in response to $M$ documented possible exceptions
• `Callee()`’s response depends normally on complex inter-relationships in persistently stored data
• It’s too hard – and unreliable – to mock up the data to trigger all $N$ responses and $M$ exceptions
Mock objects

- So instead we simply mock up each distinct response type that `Callee()` can produce.
- We do this in `Callee()` itself.
- We use conditional compilation to select the desired mock response or the normal production implementation.
- Notice that I don’t need to show you any code to explain the value of PL/SQL conditional compilation in this use case – and the best practice that is now newly supported.
Prototyping

- You often realize that more than one approach to the design of a subprogram will result in its correct behavior.
- Sometimes the alternative approaches result in source code versions which are textually largely the same but which differ critically in small areas distributed fairly evenly thought the source.
Prototyping

• For example, index by pls_integer vs index by varchar2
  • the declarations differ
  • some of the assignments might differ
  • The idiom for traversal is the same

```sql
declare
  Idx  Idx_t := Sparse_Collection.First();
begin
  while Idx is not null loop
    ...
    Idx := Sparse_Collection.Next(Idx);
  end loop;
end;
```
Prototyping

• PL/SQL conditional compilation allows all the approaches to be coded in a single source text…
• …of course, only while they are being evaluated
• It thereby eliminates the risk of carelessly introduced unintended differences
Spanning different releases of Oracle Database with a single source code corpus

- Each new release of Oracle Database brings new functionality in PL/SQL and in SQL along with new syntax for it
- Code which use a new feature won’t compile in earlier releases because of the new syntax
- ISVs typically maintain only a single source code corpus
Single code corpus spanning many releases of Oracle

- So PL/SQL code – and the SQL it contains – is written to compile in the *earliest* release of Oracle Database that the ISV supports.
- New features are not taken advantage of until usually two releases of Oracle Database after their introduction.
- Customers of the ISV who *do* use the latest release of Oracle Database are penalized by the procrastination of those who do not.
Spanning Oracle releases

$if DBMS_Db_Version.Ver_LE_9_2$ then
  declare k Index_t := 1; j Index_t := Sparse.First();
  begin
    while j is not null loop
      Dense(k) := Sparse(j);
      j := Sparse.Next(j);
      k := k + 1;
    end loop;
  end;
  forall j in 1..Dense.Last()
    insert into Tbl values Dense(j);
$else$
  forall j in indices of Sparse
    insert into Tbl values Sparse(j);
$end$

Won’t compile before 10.1
Hang on... this is paradoxical!

- PL/SQL conditional compilation is new in 10.2
- But the slide shows its use in 9.2!
- Yes, well... it was just a technique to show the potential value of conditional compilation in this use case
- It’s a latent benefit: ISVs won’t enjoy it until the next but one release after 10.2
- Sad, eh? ‘Specially ‘cos this use case motivated the feature!
Actually, I lied

- We did a ton of work to build this feature…
- …to satisfy a strongly voiced enhancement request
- It would’ve been crazy to ask folks to tolerate that level of deferred gratification that the last slide implied
- So we made conditional compilation available in 10.1.0.4…
- …and in 9.2.0.6
- There are some functionality restrictions
Relax…

- In 9.2.0.6 it’s disabled. You have to use an underscore parameter to enable it
- In 10.1.0.4 – while it is enabled by default – you can disable it by using the same underscore parameter
- In 10.2 this underscore parameter is obsolete
- You can set the underscore parameter only under the guidance of Oracle Support
- This will be allowed only for the ISV release spanning use case
Use a `ccflag` when...

- You’re selecting code you want at development time and don’t want in production (e.g. tracing and assertions)
- You want to control the conditionalization of just one compilation unit
- Rely on the fact that an undefined flag evaluates to null to write the test
Use a pkg constant when...

- You want different conditionalizations at different installation sites or at different times at the same site
- Something in the environment can be used to recompile the controlling package with appropriate [new] values of the controlling constants
The whitepaper

• I submitted just the abstract for the OpenWorld proceedings
• This provides a link to the whitepaper’s canonical location
• I can revise it periodically
• It’s there now – here’s a lightning tour
• I’ll wire up the OTN navigation ASAP
• Read it!