Advanced PL/SQL Instrumentation

Sample Debug Session

Anthony Harper

Senior Technical Architect

information architects
designing
enduring
quality

Debug: Start Block

```
begin
  -- Call the procedure
  my_package.my_other_procedure(:p_input, :p_user);
  -- Call the procedure again
  my_package.my_traced_procedure(:p_input, :p_user);
end;
```

Debug: Call Procedure 1

```
begin
  -- Call the procedure
  my_package.my_other_procedure(:p_input, :p_user);
  -- Call the procedure again
  my_package.my_traced_procedure(:p_input, :p_user);
end;
```

Debug: Package Specification

```
create or replace package my_package is

--Expose the logging object
   logger app#log_type := app#log_type('my_package');
…
```

Debug: Logger Declaration

```
create or replace package my_package is

--Expose the logging object
   logger app#log_type := app#log_type('my_package');
…
```

Debug: Type Constructor

```
constructor function app#log_type(p_object_name in varchar2)
return self as result is
   begin
      self.object_name       := p_object_name;
      self.method_name       := null;
      self.enable_exceptions := 0;
      self.enable_traces     := 0;
      self.enable_audits     := 0;
      return;
   end app#log_type;
```

Debug: Begin Constructor

```
constructor function app#log_type(p_object_name in varchar2)
return self as result is
   begin
      self.object_name       := p_object_name;
      self.method_name       := null;
      self.enable_exceptions := 0;
      self.enable_traces     := 0;
      self.enable_audits     := 0;
      return;
   end app#log_type;
```

Debug: Set Property

```
constructor function app#log_type(p_object_name in varchar2)
return self as result is
   begin
      self.object_name       := p_object_name;
      self.method_name       := null;
      self.enable_exceptions := 0;
      self.enable_traces     := 0;
      self.enable_audits     := 0;
      return;
   end app#log_type;
```

Debug: Default Properties

```
constructor function app#log_type(p_object_name in varchar2)
return self as result is
   begin
      self.object_name       := p_object_name;
      self.method_name       := null;
      self.enable_exceptions := 0;
      self.enable_traces     := 0;
      self.enable_audits     := 0;
      return;
   end app#log_type;
```

Debug: Constructor Return

```
constructor function app#log_type(p_object_name in varchar2)
return self as result is
   begin
      self.object_name       := p_object_name;
      self.method_name       := null;
      self.enable_exceptions := 0;
      self.enable_traces     := 0;
      self.enable_audits     := 0;
      return;
   end app#log_type;
```

Debug: Exit Constructor

```
constructor function app#log_type(p_object_name in varchar2)
return self as result is
   begin
      self.object_name       := p_object_name;
      self.method_name       := null;
      self.enable_exceptions := 0;
      self.enable_traces     := 0;
      self.enable_audits     := 0;
      return;
   end app#log_type;
```

Debug: Done Specification

```
create or replace package my_package is

…

end my_package;
```

Debug: Package Body

```
create or replace package body my_package is

…

begin
   -- Initialization, set up the logging state
   logger.init;
end my_package;
```

Debug: Initialize Package

```
create or replace package body my_package is

…

begin
   -- Initialization, set up the logging state
   logger.init;
end my_package;
```

Debug: Initialize Logger

```
create or replace package body my_package is

…

begin
   -- Initialization, set up the logging state
   logger.init;
end my_package;
```

Debug: Enter Type Init

```
member procedure init is
      v_errors     app#types.switch := false;
      v_traces     app#types.switch := false;
      v_audits     app#types.switch := false;
begin
get_env(v_errors, v_traces, v_audits);
self.enable_exceptions := app#types.switch_to_flag(v_errors);
self.enable_traces     := app#types.switch_to_flag(v_traces);
self.enable_audits     := app#types.switch_to_flag(v_audits);
end init;
```

Debug: Begin Init

```
member procedure init is
      v_errors     app#types.switch := false;
      v_traces     app#types.switch := false;
      v_audits     app#types.switch := false;
begin
get_env(v_errors, v_traces, v_audits);
self.enable_exceptions := app#types.switch_to_flag(v_errors);
self.enable_traces     := app#types.switch_to_flag(v_traces);
self.enable_audits     := app#types.switch_to_flag(v_audits);
end init;
```

Debug: Call get_env

```
member procedure init is
      v_errors     app#types.switch := false;
      v_traces     app#types.switch := false;
      v_audits     app#types.switch := false;
begin
get_env(v_errors, v_traces, v_audits);
self.enable_exceptions := app#types.switch_to_flag(v_errors);
self.enable_traces     := app#types.switch_to_flag(v_traces);
self.enable_audits     := app#types.switch_to_flag(v_audits);
end init;
```

Debug: Enter get_env

```
member procedure get_env
  (
     p_enable_exceptions out boolean,
     p_enable_traces     out boolean,
     p_enable_audits     out boolean
  ) is
  begin
     app#log_util.get_package_environment(
              self.object_name,
              p_enable_exceptions,
              p_enable_traces,
              p_enable_audits);
  end get_env;
```

Debug: Lookup Environment

```
member procedure get_env
  (
     p_enable_exceptions out boolean,
     p_enable_traces     out boolean,
     p_enable_audits     out boolean
  ) is
  begin
     app#log_util.get_package_environment(
              self.object_name,
              p_enable_exceptions,
              p_enable_traces,
              p_enable_audits);
  end get_env;
```

Debug: Exit get_env

```
member procedure get_env
  (
     p_enable_exceptions out boolean,
     p_enable_traces     out boolean,
     p_enable_audits     out boolean
  ) is
  begin
     app#log_util.get_package_environment(
              self.object_name,
              p_enable_exceptions,
              p_enable_traces,
              p_enable_audits);
  end get_env;
```

Debug: Set Property

```
member procedure init is
      v_errors     app#types.switch := false;
      v_traces     app#types.switch := false;
      v_audits     app#types.switch := false;
begin
get_env(v_errors, v_traces, v_audits);
self.enable_exceptions := app#types.switch_to_flag(v_errors);
self.enable_traces     := app#types.switch_to_flag(v_traces);
self.enable_audits     := app#types.switch_to_flag(v_audits);
end init;
```

Debug: Set Property

```
member procedure init is
      v_errors     app#types.switch := false;
      v_traces     app#types.switch := false;
      v_audits     app#types.switch := false;
begin
get_env(v_errors, v_traces, v_audits);
self.enable_exceptions := app#types.switch_to_flag(v_errors);
self.enable_traces     := app#types.switch_to_flag(v_traces);
self.enable_audits     := app#types.switch_to_flag(v_audits);
end init;
```

Debug: Set Property

```
member procedure init is
      v_errors     app#types.switch := false;
      v_traces     app#types.switch := false;
      v_audits     app#types.switch := false;
begin
get_env(v_errors, v_traces, v_audits);
self.enable_exceptions := app#types.switch_to_flag(v_errors);
self.enable_traces     := app#types.switch_to_flag(v_traces);
self.enable_audits     := app#types.switch_to_flag(v_audits);
end init;
```

Debug: Exit Init Method

```
member procedure init is
      v_errors     app#types.switch := false;
      v_traces     app#types.switch := false;
      v_audits     app#types.switch := false;
begin
get_env(v_errors, v_traces, v_audits);
self.enable_exceptions := app#types.switch_to_flag(v_errors);
self.enable_traces     := app#types.switch_to_flag(v_traces);
self.enable_audits     := app#types.switch_to_flag(v_audits);
end init;
```

Debug: Exit Package Initialize

```
create or replace package body my_package is

…

begin
   -- Initialization, set up the logging state
   logger.init;
end my_package;
```

Debug: Enter Procedure

```
procedure my_other_procedure
  (
     p_input in varchar2,
     p_user  in varchar2
  ) is
  begin
     logger.trace('Entered Procedure', 'my_other_procedure');
     logger.trace('Input was: ' || p_input);
     logger.trace('User was: ' || p_user);
…
```

Debug: Begin Procedure

```
procedure my_other_procedure
  (
     p_input in varchar2,
     p_user  in varchar2
  ) is
  begin
     logger.trace('Entered Procedure', 'my_other_procedure');
     logger.trace('Input was: ' || p_input);
     logger.trace('User was: ' || p_user);
…
```

Debug: Trace Entrypoint

```
procedure my_other_procedure
  (
     p_input in varchar2,
     p_user  in varchar2
  ) is
  begin
     logger.trace('Entered Procedure', 'my_other_procedure');
     logger.trace('Input was: ' || p_input);
     logger.trace('User was: ' || p_user);
…
```

Debug: Enter Trace Method

```
member procedure trace
  (
     p_message in varchar2,
     p_method  in varchar2 default '#USE_CURRENT'
  ) is
  begin
     if self.enable_traces = 1 then
        set_object_method(p_method);
        app#log_util.write_trace(
              self.object_name,
              self.method_name,
              p_message);
     end if;
  end trace;
```

Debug: Check Tracing State

```
member procedure trace
  (
     p_message in varchar2,
     p_method  in varchar2 default '#USE_CURRENT'
  ) is
  begin
     if self.enable_traces = 1 then
        set_object_method(p_method);
        app#log_util.write_trace(
              self.object_name,
              self.method_name,
              p_message);
     end if;
  end trace;
```

Debug: VALUE = Optional Trace

```
member procedure trace
  (
     p_message in varchar2,
     p_method  in varchar2 default '#USE_CURRENT'
  ) is
  begin
     if self.enable_traces = 1 then
        set_object_method(p_method);
        app#log_util.write_trace(
              self.object_name,
              self.method_name,
              p_message);
     end if;
  end trace;
```

Debug: End Check State

```
member procedure trace
  (
     p_message in varchar2,
     p_method  in varchar2 default '#USE_CURRENT'
  ) is
  begin
     if self.enable_traces = 1 then
        set_object_method(p_method);
        app#log_util.write_trace(
              self.object_name,
              self.method_name,
              p_message);
     end if;
  end trace;
```

Debug: Exit Trace Method

```
member procedure trace
  (
     p_message in varchar2,
     p_method  in varchar2 default '#USE_CURRENT'
  ) is
  begin
     if self.enable_traces = 1 then
        set_object_method(p_method);
        app#log_util.write_trace(
              self.object_name,
              self.method_name,
              p_message);
     end if;
  end trace;
```

Debug: Trace Parameters

```
procedure my_other_procedure
  (
     p_input in varchar2,
     p_user  in varchar2
  ) is
  begin
     logger.trace('Entered Procedure', 'my_other_procedure');
     logger.trace('Input was:  ' || p_input);
     logger.trace('User was:  ' || p_user);
…
```

Debug: Trace Process Point

```
 procedure my_other_procedure
   (
      p_input in varchar2,
      p_user  in varchar2
   ) is
   begin
…
      logger.trace('User was: ' || p_user);
      --do something
      logger.trace('Calling my_traced_procedure');
      my_traced_procedure(p_input, p_user);
…
```

Debug: Call Other Code

```
 procedure my_other_procedure
   (
      p_input in varchar2,
      p_user  in varchar2
   ) is
   begin
…
      logger.trace('User was: ' || p_user);
      --do something
      logger.trace('Calling my_traced_procedure');
      my_traced_procedure(p_input, p_user);
      --after sub call, reset the method name
      logger.trace('Done sub call', 'my_other_procedure');
…
```

Debug: Return From Other Code

```
 procedure my_other_procedure
…
      my_traced_procedure(p_input, p_user);
      --after sub procedure call, reset the method name
      logger.trace('Done sub call', 'my_other_procedure');
…
```

Debug: Trace Done Procedure

```
 procedure my_other_procedure
…
      --after sub procedure call, reset the method name
      logger.trace('Done sub call', 'my_other_procedure');
      logger.trace('Finished executable section');
   exception
      when others then
         logger.err(sqlerrm, 'my_other_procedure');
   end my_other_procedure;
```

Debug: Exit Procedure 1

```
 procedure my_other_procedure
…
      --after sub procedure call, reset the method name
      logger.trace('Done sub call', 'my_other_procedure');
      logger.trace('Finished executable section');
   exception
      when others then
         logger.err(sqlerrm, 'my_other_procedure');
   end my_other_procedure;
```

Debug: Call Procedure 2

```
begin
  -- Call the procedure
  my_package.my_other_procedure(:p_input, :p_user);
 -- Call the procedure again
 my_package.my_traced_procedure(:p_input, :p_user);
end;
```

Debug: Viola, No Initialization

```
procedure my_traced_procedure
  (
     p_input in varchar2,
     p_user  in varchar2
  ) is
  begin
     logger.trace('Entered Proc', 'my_traced_procedure');
     logger.trace('Input was: ' || p_input);
…
```

Replay: Viola, No Initialization

```
procedure my_traced_procedure
  (
     p_input in varchar2,
     p_user  in varchar2
  ) is
  begin
     logger.trace('Entered Proc', 'my_traced_procedure');
     logger.trace('Input was: ' || p_input);
…
```

Debug: Begin Procedure 2

```
procedure my_traced_procedure
  (
     p_input in varchar2,
     p_user  in varchar2
  ) is
  begin
     logger.trace('Entered Proc', 'my_traced_procedure');
     logger.trace('Input was: ' || p_input);
…
```

Debug: Trace Entrypoint

```
procedure my_traced_procedure
  (
     p_input in varchar2,
     p_user  in varchar2
  ) is
  begin
     logger.trace('Entered Proc', 'my_traced_procedure');
     logger.trace('Input was: ' || p_input);
…
```

Debug: Trace Parameters

```
 procedure my_traced_procedure
…
      logger.trace('Input was: ' || p_input);
      logger.trace('User was: ' || p_user);
      --do something
      logger.trace('Finished executable section');
   exception
      when others then
         logger.err(sqlerrm, 'my_traced_procedure');
   end my_traced_procedure;
```

Debug: Trace Done Procedure

```
 procedure my_traced_procedure
…
      logger.trace('Input was: ' || p_input);
      logger.trace('User was: ' || p_user);
      --do something
      logger.trace('Finished executable section');
   exception
      when others then
         logger.err(sqlerrm, 'my_traced_procedure');
   end my_traced_procedure;
```

Debug: Exit Procedure 2

```
 procedure my_traced_procedure
…
      logger.trace('Input was: ' || p_input);
      logger.trace('User was: ' || p_user);
      --do something
      logger.trace('Finished executable section');
   exception
      when others then
         logger.err(sqlerrm, 'my_traced_procedure');
   end my_traced_procedure;
```

Debug: Done

```
begin
   -- Call the procedure
  my_package.my_other_procedure(:p_input, :p_user);
 -- Call the procedure again
 my_package.my_traced_procedure(:p_input, :p_user);
end;
```

Resources

- The notes refer to the appropriate script.
- Scripts can be downloaded at:

http://plsqlarchitects.com/logging

- Feel free to email me with questions:

anthony.harper@corporateinformationarchitects.com