



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

# PLUTO - PL/SQL Unit Testing for Oracle

Josh McAdams

OSCON 2008



◆  
P  
L  
U  
T  
O  
◆

- Unit testing framework for PL/SQL
- Loosely based on the xUnit design
- Allows for an object oriented approach to testing Oracle code
- Performs tests within the database

# Why do I even need it?



◆  
P  
L  
U  
T  
O  
◆

- It feels better to do your testing in the language that you are coding in.
- PL/SQL is becoming more and more popular for writing feature-rich programs.



◆  
P  
L  
U  
T  
O  
◆

```
declare
    ut_obj my_test_obj := my_test_obj( );
begin
    ut_obj.run_tests;
end;
```



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

# Order of Method Execution



# Building a Testing Object



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

```
create or replace type testing_obj
  under pluto_obj (
    constructor function testing_obj
      return self as result
  )
  instantiable not final;
```



# Setting Up Utility and Output Objects

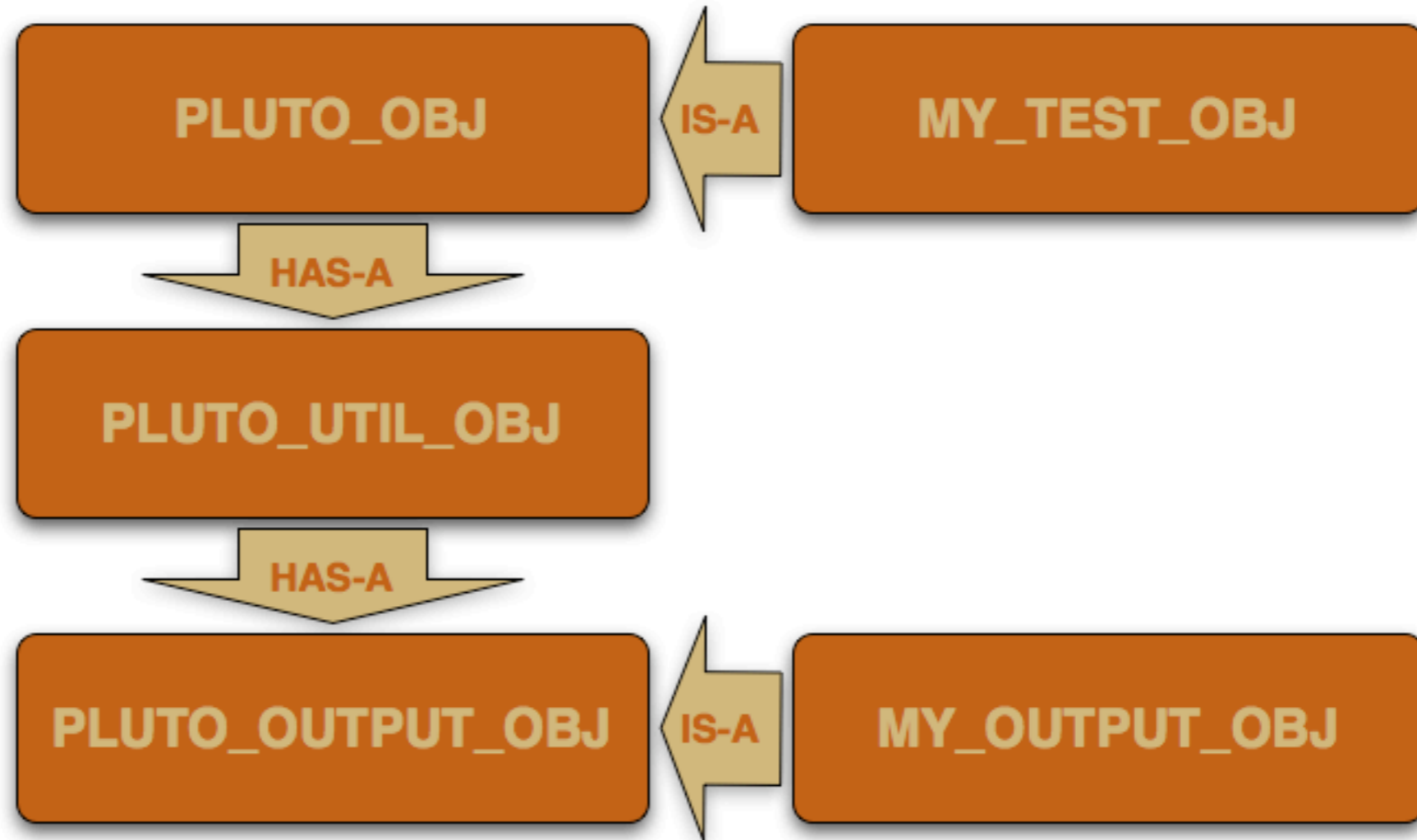
◆  
P  
L  
U  
T  
O

```
create or replace type body testing_obj is
  constructor function testing_obj
    return self as result is
begin
  m_util_object :=
    pluto_util_obj(
      output_object =>
        pluto_output_obj( )
    );
  return;
end testing_obj;
end;
```



◆  
P  
L  
U  
T  
O  
◆

# Class Relationships





◆  
P  
L  
U  
T  
O  
◆

# PLUTO\_OUTPUT\_OBJ

```
create or replace type pluto_output_obj
  authid current_user as object (
  ...
  constructor function pluto_output_obj
    return self as result,
--
  member procedure log_test_count( test_count in number ),
--
  member procedure log_test_results(
    test_label    in varchar,
    test_passed   in boolean,
    details       in varchar := ''
  ),
--
  member procedure log_message( message in varchar := '' ),
--
  member procedure log_test_completion
--
)
instantiable not final;
```



# PLUTO\_OUTPUT\_OBJ

◆  
P  
L  
U  
T  
O  
◆

```
Test Count [2]
testing message one
1 - test one passed
2 - test two failed
test message two of more than one
1 tests passed
1 tests failed
50 percent of expected tests successful
```



# PLUTO\_OUTPUT\_TAP\_OBJ

◆  
P  
L  
U  
T  
O  
◆

```
create or replace type pluto_output_tap_obj
  authid current_user under pluto_output_obj(
--
  constructor function pluto_output_tap_obj
    return self as result,
--
  overriding member procedure log_test_count( test_count number ),
--
  overriding member procedure log_test_results(
    test_label   in varchar,
    test_passed  in boolean,
    details      in varchar := ''
  ),
--
  overriding member procedure
    log_message( message in varchar := '' ),
--
  overriding member procedure log_test_completion
--
)
instantiable not final;
```



# PLUTO\_OUTPUT\_TAP\_OBJ

◆  
P  
L  
U  
T  
O  
◆

```
1..2
# testing message one
ok - test one
not ok - test two
# test message two of more than one
```



# PLUTO\_OUTPUT\_TAP\_OBJ

◆  
P  
L  
U  
T  
O  
◆

```
--(0)> prove my_tests.t  
my_tests....FAILED test 2  
          Failed 1/2 tests, 50.00% okay  
Failed Test Stat Wstat Total Fail  Failed  List of Failed  
-----  
my_tests.t                2     1 50.00%  2  
Failed 1/1 test scripts, 0.00% okay. 1/2 subtests failed, 50.00% okay.
```



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

# PLUTO\_UTIL\_OBJ

```
CREATE OR REPLACE type pluto_util_obj authid current_user as object(
  _output_obj pluto_output_obj,
  --
  constructor function pluto_util_obj(
    output_object in pluto_output_obj := null
  )
  return self as result,
  --
  member procedure set_test_count(test_count in number),
  --
  member procedure finish,
  --
  member procedure log(message in varchar),
  --
  member function ok(
    self in out pluto_util_obj,
    test_passed in boolean,
    test_label in varchar
  )
  return boolean,
  --
  member procedure ok(test_passed in boolean, test_label in varchar),
  --
  member function is_ok(
    self in out pluto_util_obj,
    data_got in anydata,
    data_expected in anydata,
    test_label in varchar
  )
  return boolean,
  --
  member procedure is_ok(
    data_got in anydata,
    data_expected in anydata,
    test_label in varchar
  ),
  --
  member function is_ok(
    self in out pluto_util_obj,
    data_got in number,
    data_expected in number,
    test_label in varchar
  )
  return boolean,
  --
  member procedure is_ok(
    data_got in number,
    data_expected in number,
    test_label in varchar
  ),
  --
  member function is_ok(
    self in out pluto_util_obj,
    data_got in varchar,
    data_expected in varchar,
    test_label in varchar
  )
  return boolean,
  --
  member procedure is_ok(
    data_got in varchar,
    data_expected in varchar,
    test_label in varchar
  ),
  --
  member function is_ok(
    self in out pluto_util_obj,
    data_got in boolean,
    data_expected in boolean,
    test_label in varchar
  )
  return boolean,
  --
  member procedure is_ok(
    data_got in boolean,
    data_expected in boolean,
    test_label in varchar
  ),
  --
  member function is_ok(
    self in out pluto_util_obj,
    data_got in date,
    data_expected in date,
    test_label in varchar
  )
  return boolean,
  --
  member procedure is_ok(
    data_got in date,
    data_expected in date,
    test_label in varchar
  ),
  --
  member function is_ok(
    self in out pluto_util_obj,
    data_got in timestamp,
    data_expected in timestamp,
    test_label in varchar
  )
  return boolean,
  --
  member procedure is_ok(
    data_got in timestamp,
    data_expected in timestamp,
    test_label in varchar
  ),
  --
  member function is_ok_helper(
    self in out pluto_util_obj,
    test_passed in boolean,
    data_got in varchar,
    data_expected in varchar,
    test_label in varchar
  )
  return boolean,
  --
  member function table_exists(
    self in out pluto_util_obj,
    table_name in varchar
  )
  return boolean,
  --
  member procedure table_exists(table_name in varchar)
  --
)
instantiable not final;
```



◆  
P  
L  
U  
T  
O  
◆

# PLUTO\_UTIL\_OBJ

```
CREATE OR REPLACE type pluto_util_obj
  authid current_user as object(
    m_output_obj pluto_output_obj,
  --
    constructor function pluto_util_obj(
      output_object in pluto_output_obj := null
    )
      return self as result,
  --
    member procedure set_test_count(test_count in number),
  --
    member procedure finish,
  --
    member procedure log(message in varchar),
  --
    ...
  )
instantiable not final;
```



◆  
P  
L  
U  
T  
O  
◆

# PLUTO\_UTIL\_OBJ

```
...
member function ok(
    self          in out  pluto_util_obj,
    test_passed   in      boolean,
    test_label    in      varchar
)
    return boolean,
--
member procedure
    ok(test_passed in boolean, test_label in varchar),
...
```



◆  
P  
L  
U  
T  
O

# PLUTO\_UTIL\_OBJ

```
...
member function is_ok(
    self          in out  pluto_util_obj,
    data_got      in      anydata,
    data_expected in      anydata,
    test_label    in      varchar
)
    return boolean,
--
member procedure is_ok(
    data_got      in      anydata,
    data_expected in      anydata,
    test_label    in      varchar
),
...
```

Overridden for anydata, number, varchar, boolean, date, timestamp



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

# PLUTO\_UTIL\_OBJ

```
1..4
# testing message one
ok - test one
not ok - test two
ok - ok number
not ok - not ok number
# Got '42' when expected '40'
ok - ok varchar
not ok - not ok varchar
# Got 'a' when expected 'b'
ok - ok boolean
not ok - not ok boolean
# Got 'TRUE' when expected 'FALSE'
ok - ok date
not ok - not ok date
# Got '20081022010204' when expected '20081022010203'
ok - ok timestamp
not ok - not ok timestamp
# Got '20081022010204123456000' when expected '20081022010203123456000'
not ok - x exists
```



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

# PLUTO\_UTIL\_OBJ

```
...  
member function table_exists(  
    self          in out  pluto_util_obj,  
    table_name    in      varchar  
)  
    return boolean,  
--  
member procedure table_exists(table_name in varchar)  
...
```



◆  
P  
L  
U  
T  
O  
◆

- Executed once per test run
- Executed in order of the current sort order in the database
- Can assert any number of test cases



◆  
P  
L  
U  
T  
O  
◆

# TEST% Procedures

```
member procedure test_one is
  v_i number;
begin
  execute immediate 'select i from x' into v_i;
  m_util_object.is_ok(
    data_got      => v_i,
    data_expected => 1,
    test_label    => 'checking value of one'
  );
end test_one;
```



# STARTUP% Procedures

◆  
P  
L  
U  
T  
O

- Executed once per test suite run.
- If there are multiple startup procedures they are executed in the order that the sort based on the current database settings.
- Good place to initialize test count and to set up environment by creating tables, etc.

# STARTUP% Procedures



◆  
P  
L  
U  
T  
O  
◆

```
member procedure startup_testing is
begin
    m_util_object.set_test_count( 1 );
    execute immediate 'create table x ( i number )';
end startup_testing;
```

# SHUTDOWN% Procedures



◆  
P  
L  
U  
T  
O  
◆

- Executed once per test suite run.
- If there are multiple shutdown procedures they are executed in the order that the sort based on the current database settings.
- Good place to call finish on the PLUTO utility object and to do a final cleanup on your testing environment.

# SHUTDOWN% Procedures



◆  
P  
L  
U  
T  
O  
◆

```
member procedure shutdown_testing is
begin
    execute immediate 'drop table x';
    m_util_object.finish;
end shutdown_testing;
```



◆  
P  
L  
U  
T  
O  
◆

- Executed once before each TEST% procedure.
- If there are multiple setup procedures they are executed in the order that the sort based on the current database settings.
- This is a good place to seed data that needs to be reset for each test.



# SETUP% Procedures

◆  
P  
L  
U  
T  
O  
◆

```
member procedure setup_testing is
begin
    execute immediate
        'insert into x (i) values (1)';
end setup_testing;
```

# TEARDOWN% Procedures



◆  
P  
L  
U  
T  
O  
◆

- Executed once after each TEST% procedure.
- If there are multiple teardown procedures they are executed in the order that the sort based on the current database settings.
- This is a good place to remove data that needs to be reset for each test.

# TEARDOWN% Procedures



◆  
P  
L  
U  
T  
O  
◆

```
member procedure teardown_testing is
begin
    execute immediate
        'delete from x';
end teardown_testing;
```



◆  
P  
L  
U  
T  
O  
◆

# Running a Test Suite

```
declare
    testing    testing_obj;
begin
    testing := testing_obj( );
    testing.run_tests;
end;
```



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

# Running a Test Suite

```
1..1  
ok - checking value of one
```



# Add Another Test Procedure

◆  
P  
L  
U  
T  
O  
◆

```
member procedure test_two is
    v_i number;
begin
    execute immediate
        'select count(*) from x' into v_i;
    m_util_object.is_ok(
        data_got          => v_i,
        data_expected    => 1,
        test_label       => 'checking count of one'
    );
end test_two;
```

# Add Another Test Procedure



P

L

U

T

O



1..2

ok - checking value of one

ok - checking count of one

# Running Specific Tests

```
declare
    testing    testing_obj;
begin
    testing := testing_obj(
        named_like => 'ONE'
    );
    testing.run_tests;
end;
```



◆  
P  
L  
U  
T  
O  
◆

# Running Specific Tests



P

L

U

T

O



PL/SQL  
Unit Testing  
for Oracle

1..2

ok - checking value of one



◆  
P  
L  
U  
T  
O  
◆

- Smarter calculation of test counts
- Code coverage statistics
- Tags/Groups for tests
- A package-based equivalent
- More output types
- General optimizations
- Richer utility set



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

# Installing the System

```
SQL> create user pluto identified by pluto;
```

```
User created.
```

```
SQL> grant create session to pluto;
```

```
Grant succeeded.
```

```
SQL> grant create type to pluto;
```

```
Grant succeeded.
```

```
SQL> @install.sql
```

```
...
```

# Testing the Test Framework



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

- Cheated and used Perl
- Pretty much happy-path testing right now

# Testing the Test Framework



◆  
P  
L  
U  
T  
O  
◆

```
export PLUTO_TEST_USER=pluto  
export PLUTO_TEST_PASSWORD=pluto  
export PLUTO_TEST_SID=demo11  
perl t/tests_pulled_from_sql_file.t
```



◆  
P  
L  
U  
T  
O  
◆

PL/SQL  
Unit Testing  
for Oracle

# Where To Get PLUTO

<http://code.google.com/p/pluto-test-framework/downloads/list>