utPLSQL v3 Proposal

- Revisions
- Overview
  - Goal of utPLSQL
  - Objectives of utPLSQL v3 Core
- Some Details
  - What needs to be removed from utPLSQL?
  - Additional Add-ons
  - What is an Add-on?
- Some Examples with v3
  - Start Writing Assertion Tests Immediately
  - TestSuites
  - UTPLSQL_INIT Procedure
  - Example TestSuite Package
  - Example Session
- v3 Database Objects
  - Schema
  - Packages
- The Upgrade Path
  - Installation
  - Existing Software Modification
- Appendix A - Definitions
- Appendix B - Web Links
- Appendix C - Design Approach
  - Data Access Object Pattern
  - Database Constraint Enforcement
  - Consistent Call Hierarchy
  - SOLID Design Principles
- Appendix D - Legacy Package Disposition
- Appendix E - Legacy API Disposition
- Appendix F - Current utPLSQL Repository
  - ER Diagram
  - Tables
- Appendix G - Legacy utPLSQL Selected Call Trees
  - UTPLSQL RUN and TEST Procedures
  - UTPLSQL Initialization
  - UTPLSQL Main Execution
  - utReport and utOutputReport Packages
  - utResult Show & utResult2 Report Procedures
  - From utAssert to utOutputReporter

Revisions

Ver.	Date	Description
0.1	Unfinished	First Draft

Overview

The current version of utPLSQL is v2. This is a proposal for utPSQL version v3. The general direction is to streamline the utPSQL core and implement peripheral functionality outside the core. The streamlined core will return all functionality to the basic packages like UTPLSQL and UTASSERT, while incorporating new functionality from the v2 packages like UTPLSQL2 and UTASSERT2 into the v1 packages. Given that most all v2 features remained undocumented outside the source code, and v2 was not required for basic operation, v2 APIs are not being preserved.

Goal of utPLSQL

Provide a unit testing framework for PL/SQL developers modeled on the Junit and Xunit frameworks. (http://sourceforge.net/projects/utplsql/)

Objectives of utPLSQL v3 Core

1. Build on the utPLSQL heritage where possible.
2. Maximize flexibility
    a. PL/SQL driven testing.
    b. No required “prefixes” on names.
    c. No required setup or teardown procedures.
3. Minimize required maintenance
    a. No pre-setup of TestSuite/TestCase configuration
    b. No historical data collection
4. Improved robustness
    a. Allows for testing of "other" DOUTs like Table Triggers and non-source objects like Views.
    b. Code Coverage - Percent of source executed within a DOUT
5. Add-on Based Expansion
    a. New and former functionality in separate installations
    b. Simplified Add-on development through greater separation of concerns

Some Details

What needs to be removed from utPLSQL?

Overly complicated install/update.
Keep the installations simpler and easier to understand. PL/SQL developers should be capable of basic version control. Accordingly, different versions of utPLSQL for different versions of Oracle are expectable.

Troubled repository.
It is evident that the utPLSQL repository started life supporting the pre-configured Test Suite setup required in the early versions of utPLSQL. As later versions of utPLSQL adopted support for ad-hoc Test Suites, the data was adapted into the existing design. This adaptation makes the repository difficult to understand and use. Additionally, historical test data storage is not needed on many modern build servers that capture test results in other locations.

Too many functions.
These functions should be split into utPSQL add-ons as they are no longer required in the utPSQL core. Also, splitting these functions into Add-ons will help distribute the man-power required to support future utPLSQL development.

- **PLUS Assertions Add-on.** This add-on contains more complex assertions for specific database functionality, like DBMS_PIPES. Moving these assertions to an add-on helps to streamline the utPLSQL core.
- **PLUS Reporter Add-on**. This add-on will load v2 reporters like HTML and File.
- **Test Package Generator Add-on**. This add-on will generate test skeletons for use as a starting point for a unit test package.
- **Structure Comparison Generator Add-on**. This add-on is Dan Spencer's contribution that creates functions for the comparison of record types.

Additional Add-ons

- **Jenkins JUnit XML Add-on**. This add-on generates JUnit XML files compatible with Jenkins from the unit test data of the last TestSuite execution.
- **Historical Storage Add-on**. This add-on implements the concept of a "test run". As a test suite is completed, this add-on captures the test data for historical storage in a simplified schema.
- **Automatic Source Recompile Add-on**. This add-on will perform the automatic recompile from a database file system directory.
- **Legacy Repository Add-on**. This add-on will load unit test data from utPLSQL v3 data structures to the legacy repository. This add-on will also add more v2 compatible APIs.

What is an Add-on?

An add-on is software which enhances another software application and usually cannot be run independently. utPLSQL add-ons rely on the utPLSQL core and one or more other utPLSQL add-ons. Each add-on complies with these guidelines:

- Adds additional functionality to the utPLSQL core.
- Runs entirely in the Oracle database.
- Installs database objects in the same schema as the utPLSQL core.
- Maintains version compatibility with core and compatible add-ons.

Some Examples with v3

Start Writing Assertion Tests Immediately

This is what most developers want. Unit testing that can start immediately, anywhere, by running one of the test procedures in the UTASSERT package. The results of the test are sent to DBMS_OUTPUT for display. Following is an example test.

```
declare
   var1  dual.dummy%TYPE;
begin
   select dummy into var1 from dual;
   utassert.eq(msg_in          => 'Testing Dual Table Contents'
              ,check_this_in   => var1
              ,against_this_in => 'X');
end;
/

SUCCESS - : EQ "Testing Dual Table Contents" Expected "X" and got "X"
>
```

TestSuites

Configuring a TestSuite is as simple as creating an Oracle Database Package. There are a few objects in the package that have special meaning:

- **Any procedure that has no parameters** - These are treated as TestCases. They are executed in alphanumeric order.
- **A UTPLSQL_INIT procedure with no parameters** - This is used for additional TestSuite setup, like identifying the Database Object Under Test.
- **A UTPLSQL_FIN procedure with no parameters** - Reserved for future use.

To execute a TestSuite package, use this call.

```
procedure utplsql.run(i_package_name   varchar2
                     ,i_package_owner  varchar2)
```

- i_Package_Name - Name of the TestSuite package.
- i_Package_Owner - Optional name of the TestSuite package owner.

UTPLSQL_INIT Procedure

This is a new procedure that allows configuration of a Test Suite within the Test Suite package. The following items are TestSuite specific and are configured using special calls within this procedure.

utplsql.set_dout

This procedure sets the DOUT (Database Object Under Test) and the RunCodeCoverage Flag.

Example TestSuite Package

```
create or replace package my_testsute1 as
   procedure utplsql_init;
   procedure utplsql_fin;
   procedure my_testcase1;
   procedure my_testcase2;
end my_testsuite1;
/
create or replace package body my_testsuite1 as
----------------------------------------
procedure utplsql_init as
begin
   utplsql.set_dout(dout_name_in       => 'TARGET_PACKAGE'
                    dout_type_in       => 'PACKAGE'
                    dout_schema_in     => 'OTHER_SCHEMA'
                    RunCodeCoverage_in => 'Y');
end utplsql_init;
----------------------------------------
procedure utplsql_fin as
begin
   null;
end utplsql_fin;
----------------------------------------
procedure my_testcase1 as
begin
   utassert.eq(msg_in          => 'Sample Test1'
               ,check_this_in   => 'Yes'
               ,against_this_in => 'Yes');
end my_testcase1;
----------------------------------------
procedure my_testcase2 as
begin
   assert.eq(msg_in          => 'Sample Test1'
             ,check_this_in   => 'No'
             ,against_this_in => 'No');
end my_testcase2;
----------------------------------------
end my_testsuite1;
/
```

Example Session

1. Start Oracle Session
2. Run One or More TestSuites
3. Review DBMS_OUTPUT results
4. End Oracle Session

v3 Database Objects

Schema

These tables are implemented as temporary tables to reduce administrative overhead and improve test execution performance. These temporary tables are set to clear their contents at the end of the database session.

UTC_TESTSUITE Table

This table stores data about the Test Suites that have been executed during the current session. Overall runtime, overall test results, and overall errors are recorded here.

UTC_TESTCASE Table

This table stores data about the Test Cases that have been executed within each Test Suite. Test Case runtime, Test Case results, and errors caught during each Test Case are recorded here.

UTC_OUTCOME Table

This table stores data about outcomes of individual assertions executed during each Test Case. Time of execution, assertion result, descriptions, messages, and errors caught during the assertion are recorded here.

Packages

UTPLSQL3 Package

This is the main execution package. It has a new name to reflect the fundamental change in testing procedures from the v1 and v2 versions.

UTASSERT Package

The UTASSERT package contains the assertions. Each assertion calls the "this" assertion. The "this" assertion follows this form:

1. If UTPLSQL RunCodeCoverage then Pause Profiler
2. Execute Assertion
    - Trap Exceptions
    - Create Results
3. Call UTPLSQL Results
4. If UTPLSQL RunCodeCoverage, Resume Profiler

UTASSERT2 Package

The UTASSERT2 package is included for legacy compatibility. All of the assertions in UTASSERT2 call assertions in the UTASSERT package. (This is the reverse of the v2 implementation.)

UTOUTPUTREPORTER Package

This is a required reporter as it is compiled into the UTPLSQL package. It is also the default reporter if a custom reporter fails.

The Upgrade Path

These are the modifications required to upgrade existing utPSQL installations to utPLSQL v3.

Installation

The extreme reduction in functionality allows the v3 database objects to be installed directly into the schema of the previous utPLSQL installation. A "Legacy Repository Add-on" can be used to continue data population of the previous utPSQL schema.

Existing Software Modification

Appendix A - Definitions

"If you can't explain it simply, you don't understand it well enough." - Albert Einstein

XUnit Definitions
https://en.wikipedia.org/wiki/XUnit (Includes minor editing for clarification)

- **Test runner**- An executable program that runs tests implemented using an xUnit framework and reports the test results.
- **Test case** - The most elemental class. All unit tests are inherited from here.
- **Test fixtures** - (also known as a test context) The set of preconditions or state needed to run a test. The developer should set up a known good state before the tests, and return to the original state after the tests.
- **Test suites** - Set of tests that all share the same **test fixture**. The order of the tests shouldn't matter.
- **Test execution** - The execution of an individual unit test including:
    - **Setup** - First, we should prepare our 'world' to make an isolated environment for testing
    - **Body of test** - Here we make all the tests
    - **Teardown** - At the end, whether we succeed or fail, we should clean up our 'world' to not disturb other tests or code
    - The **setup** and **teardown** serve to initialize and clean up **test fixtures**.
- **Test result formatter** - Produces results in one or more output formats. In addition to a plain, human-readable format, there is often a test result formatter that produces XML output. The XML test result format originated with JUnit but is also used by some other xUnit testing frameworks, for instance build tools such as Jenkins and Atlassian Bamboo.
- **Assertions** - A function or macro that verifies the behavior (or the state) of the unit under test. Usually an assertion expresses a logical condition that is true for results expected in a correctly running system under test (SUT). Failure of an assertion typically throws an exception, aborting the execution of the current test.

Java Definitions

- **Object** - In computer science, an object can be a variable, a data structure, or a function, and as such, is a location in memory having a value and possibly referenced by an identifier. (https://en.wikipedia.org/wiki/Object_(computer_science))
- **Class** - In object-oriented programming, a class is an extensible program-code-template for creating **objects**, providing initial values for state (member variables) and implementations of behavior (member functions or methods) (https://en.wikipedia.org/wiki/Class_(computer_programming))

Junit Definitons
https://github.com/junit-team/junit/wiki

- **Assertion** - JUnit provides overloaded assertion methods for all primitive types and Objects and arrays (of primitives or Objects).

- **Test Runners** - JUnit provides tools to define the **suite** to be run and to display its results. To run tests and see the results on the console, run this from a Java program.
- **Suite** - Using Suite as a **test runner** allows you to manually build a suite containing tests from many **classes**.
- **Execution Order** - From version 4.11, JUnit will by default use a deterministic, but not predictable, order(MethodSorters.DEFAULT). To change the test execution order simply annotate your test class using @FixMethodOrder and specify one of the available MethodSorters
- **Test Fixture** - A test fixture is a fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well known and fixed environment in which tests are run so that results are repeatable.

Oracle Database Definitions

- **Database Object** - Listed in USER_OBJECTS. Examples include packages, types, and tables.
- **Schema** - Database owner of a **database object**.

utPLSQL v3 Definitions

- **DOUT** - Database Object Under Test. Name of the database object being tested.
- **Assertion** - Simple PASS/FAIL test.
- **TestCase** - Collection of **Assertions** within a PL/SQL package procedure. Other software included in each TestCase include setup, teardown, and intermediate setups (Test Fixtures). Assertions are part of the procedure execution. Each TestCase is custom written.
- **TestSuite** - Collection of **TestCases** with a common **DOUT** within a PL/SQL package. TestCases are executed in alphanumeric order. Each TestSuite is custom written.
- **Test runner**- The UTPLSQL.run procedure installed as part of the utPLSQL software.

JUnit XML for Jenkins Definitions

These definitions are based around the JUnit XML for Jenkins requirement. There is some translating required as the Oracle database is relational, not object oriented. Additionally, the Jenkins XML specification has some nuances that are not obvious.
(http://nelsonwells.net/2012/09/how-jenkins-ci-parses-and-displays-junit-output/)
(https://pzolee.blogs.balabit.com/2012/11/jenkins-vs-junit-xml-format/)

- **Class** - Java Unit Under Tested (UUT). In the Oracle database, this equates to a database object
- **Package** - Collection of **Classes**. In the Oracle database, this equates to a database schema.
- **Assertion** - Simple PASS/FAIL test.
- **TestCase** - Collection of **Assertions** with a common **Class**.
- **TestSuite** - Collection of **TestCases**.

Appendix B - Web Links

- http://utplsql.sourceforge.net/
- http://sourceforge.net/projects/utplsql/
- https://wiki.jenkins-ci.org/display/JENKINS/Bundled+Plugins
- https://wiki.jenkins-ci.org/display/JENKINS/JUnit+Plugin
- https://svn.jenkins-ci.org/trunk/hudson/dtkit/dtkit-format/dtkit-junit-model/src/main/resources/com/thalesgroup/dtkit/junit/model/xsd/
- https://pzolee.blogs.balabit.com/2012/11/jenkins-vs-junit-xml-format/
- http://www.tutorialspoint.com/junit/junit_suite_test.htm
- http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/d_profil.htm#sthref6972
- http://docs.oracle.com/cd/E11882_01/server.112/e41084/sql_elements008.htm#i27570

Appendix C - Design Approach

This appendix gives guidance on the design approach for implementation. While philosophical discussions are interesting and relevant, consistency demands direction. The approach listed here is that direction.

Data Access Object Pattern

The data access object is a common object oriented pattern used in PL/SQL where SQL for CRUD (Create, Retrieve, Update, and Delete) is encapsulated within a package. Packages within utPLSQL like UTPACKAGE, UTSUITE, UTTEST, and UTTESTSUITE exemplify this concept with their published methods ADD, REM, and UPD. These will be removed and replaced with SQL in each package.

Alternatively, universal use of a table API generation tool could be adopted and enforced by the utPLSQL project.

Database Constraint Enforcement

While it is possible to maintain data integrity within a relational database without them, constraints will be defined in the database as much as possible. On the rare exception that a constraint causes an unacceptable performance loss, the constraint will be disabled in the database.

Consistent Call Hierarchy

A consistent call hierarchy exists when PL/SQL programs (packages, procedures, and functions) can be listed from top to bottom and there exists no calls the occur between programs in an upward direction. Examples of inconsistencies in call hierarchy include peer calls (A calls B and B calls A) and circular calls (A calls B, B calls C, and C calls A). As much as possible, consistent call hierarchies will be used.

SOLID Design Principles

This discussion on SOLID design principles comes from Robert Love (http://robstechcorner.blogspot.com/).

- Each Package has a single focus area. ("S"OLID) It should not do too much.
- Since there is no inheritance i can't see out the Open-Close principal could be applied (S"O"LID)
- Interfaces can be developed with a implied contract that a package will have a given set of methods that have the correct parameters. "Tests", "Reporters" use this concept. (SO"L"ID)
- Interface should be be developed as needed, and be as specific as needed instead of having a single type that defines all dynamic code. Currently I have "tests" and "reporters", and like your concept. (SOL"I"D)

- Dependancies where possible should be passed in. (SOLI"D") Example: Don't select from the db if it can be passed in from another package.

Additional Design Principles from Robert Love

This discussion on SOLID design principles comes from Robert Love (http://robstechcorner.blogspot.com/).

- Avoid PackageA referencing PackageB, and PackageB referencing PackageA. I maintain code like this and it's a nightmare of spaghetti. I have found having a "types" package is critical reducing the possibility of this. Also when this occurs you typically have not followed ("S"OLID)
- DMBS_Output should only occur when user of the system asks for it.
- Our code should not impact the code that is being tested. So avoid Commit or Rollback.
- Tests should not silently fail to run. (v2 currently can happen if test packages are invalid)

Appendix D - Legacy Package Disposition

- Leg - Legacy Compatibility Add-on.
- Cmpl - Automatic Source Recompile Add-on.
- Rpt - Custom Reporter Add-on.
- Gen - Test Package Generator Add-on.
- Stct - Structure Comparison Generator Add-on.

Package Name	Core	PLUS	Leg	Rpt	Gen	Stct	Cmpl	Other
UT_UTOUTPUT								Neither called nor documented
UT_UTREPORT								Neither called nor documented
UTASSERT	X							Add UTASSERT2 functionality
UTASSERT2			X					Standardizing without Numbers
UTCONFIG			X					Discontinued: Single Responsibility
UTFILEREPORTER				X				UTL_FILE Based
UTGEN					X			
UTHTMLREPORTER				X				UTL_FILE Based
UTOUTCOME								Discontinued DAO Pattern
UTOUTPUT				X				
UTOUTPUTREPORTER				X				
UTPACKAGE								Obsolete
UTPIPE		X						DBMS_PIPE unpack
UTPLSQL	X							
UTPLSQL_UTIL		X						RefCursor Comparison
UTPLSQL2			X					Undocumented Features
UTRECEQ						X		
UTREPORT	X							Newly modified
UTRERROR								
UTRESULT								
UTRESULT2			X					

UTROUTCOME								
UTRSUITE								Discontinued DAO Pattern
UTRTESTCASE								
UTRUNITTEST								
UTRUTP								
UTSUITE								Obsolete
UTSUITEUTP								
UTTEST								
UTTESTCASE								
UTTESTPREP								Commented as No Longer Used
UTUNITTEST								
UTUTP								

```
with q1 as (
select name
      ,count(*)                       lines
 from  user_source
 where type = 'PACKAGE BODY'
 group by name
), q2 as (
select package_name                  name
      ,count(distinct subprogram_id) subprogs
      ,nvl(max(overload),0) + 1      maxovrlds
 from  user_arguments
 group by package_name
)
select q1.name
      ,q1.lines
      ,q2.subprogs
      ,q2.maxovrlds
 from q1 join q2 on (q2.name = q1.name)
 order by lines desc;

NAME              LINES  SUBPROGS  MAXOVRLDS
----------------  -----  --------  ---------
UTASSERT2         3656   64        7
UTPLSQL_UTIL      1623   28        4
UTGEN             1453   22        3
UTPLSQL           1410   27        5
UTCONFIG          762    34        1
UTASSERT          651    34        4
UTPLSQL2          475    19        1
UTPACKAGE         430    10        3
UTUTP             410    17        3
UTRESULT2         354    11        3
UTOUTPUTREPORTER  351    11        1
```

UTRERROR	337	9	1
UTRECEQ	317	3	1
UTRESULT	310	19	3
UTSUITE	280	10	3
UTFILEREPORTER	240	5	1
UTROUTCOME	233	6	3
UTRUTP	231	6	3
UTREPORT	224	14	3
UT_UTOUTPUT	218	7	1
UTUNITTEST	193	9	3
UTSUITEUTP	192	7	3
UTTEST	186	8	3
UTTESTCASE	185	8	3
UTOUTPUT	172	10	5
UTHTMLREPORTER	167	11	1
UT_UTREPORT	147	3	1
UTRUNITTEST	145	2	1
UTRSUITE	143	2	1
UTRTESTCASE	137	2	1
UTOUTCOME	133	6	3
UTPIPE	102	1	1
UT_BETWNSTR	66	3	1
UTTESTPREP	44	2	1

Appendix E - Legacy API Disposition

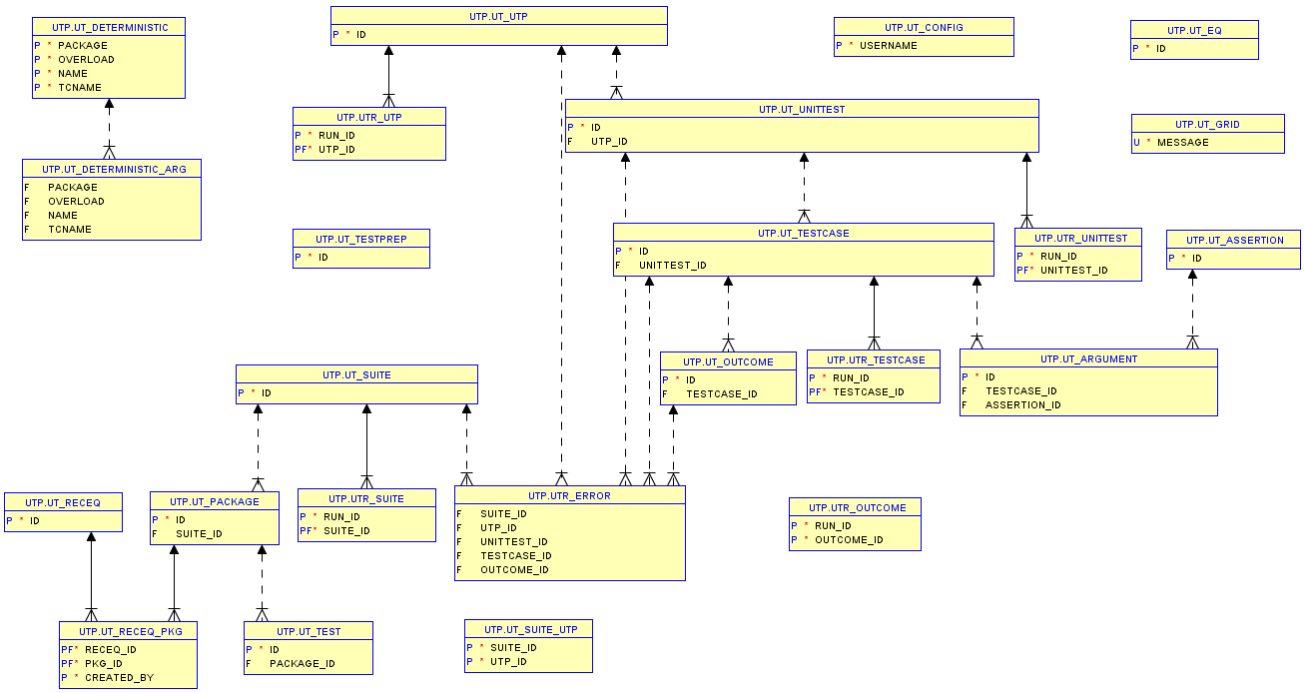
Package	Type	Name	InDoc	Notes
UTPLSQL	Function	bool2vc	No	Move to UTUTIL
UTPLSQL	Function	currpkg	No	
UTPLSQL	Function	currpkgowner	No	
UTPLSQL	Function	ifelse		Utility?
UTPLSQL	Function	ispackage	No	Move to UTUTIL
UTPLSQL	Function	pkgname		Utility?
UTPLSQL	Function	progname		Utility?
UTPLSQL	Function	seqval	No	Utility?
UTPLSQL	Function	tracing	YES	Utility?
UTPLSQL	Function	vc2bool	No	Move to UTUTIL
UTPLSQL	Function	version	YES	Utility?
UTPLSQL	Procedure	addtest	YES	Move to New Add-On
UTPLSQL	Procedure	notrc	YES	Utility?
UTPLSQL	Procedure	run	YES	
UTPLSQL	Procedure	runsuite	YES	
UTPLSQL	Procedure	test	YES	
UTPLSQL	Procedure	testsuite	YES	
UTPLSQL	Procedure	trc	YES	Utility?
UTPLSQL	Record	test_rt	No	
UTPLSQL	Record	testcase_rt	No	
UTPLSQL	Table	test_tt	No	
UTPLSQL	Variable	c_disabled	No	Constant
UTPLSQL	Variable	c_enabled	No	Constant
UTPLSQL	Variable	c_failure	No	Constant
UTPLSQL	Variable	c_no	No	Constant
UTPLSQL	Variable	c_setup	No	Constant
UTPLSQL	Variable	c_success	No	Constant
UTPLSQL	Variable	c_teardown	No	Constant

UTPLSQL	Variable	c_yes	No	Constant

Appendix F - Current utPLSQL Repository

These are the results from a fresh install of the "utplsql-2-3-0.zip" file.

ER Diagram



There are 24 database tables created during the utPLSQL installation. This E-R Diagram was created from an SQL Developer Data Modeler Data Dictionary Import of those tables.

Tables

UT_ASSERTION Table

NAME	DESCRIPTION
THIS	General Boolean expression evaluation
EQ	Scalar equality
EQTABLE	Database table/view equality
EQTABCOUNT	Database table/view row count comparison
EQQUERY	Query result set equality
EQQUERYVALUE	Query return value comparison
EQFILE	Operating system file equality
EQPIPE	Database pipe equality
EQCOLL	Collection equality - direct access
EQCOLLAPI	Collection equality - API access
ISNOTNULL	NOTNULL validation
ISNULL	NULL validation
RAISES	Raised exception validation
FILEEXISTS	Confirm that specified file exists
EVALUATE	General evaluation of specified expression
OBJEXISTS	Confirm that specified Object exists
OBJNOTEXISTS	Confirm that specified Object does not exist

The UT_ASSERTION table is pre-loaded with data when utPLSQL is installed in the database. There are 16 more columns in this table, but they all have the same data values.

UT_CONFIG Table

USERNAME	UTP
AUTOCOMPILE	N
REGISTERTEST	N
DIRECTORY	
NAMING_MODE	
PREFIX	
DELIMITER	##
SHOW_FAILURES_ONLY	N
FILEDIR	
FILEUSERPREFIX	
FILEINCPROGNAME	N
FILEDATEFORMAT	yyyyddmmhh24miss
FILEEXTENSION	.UTF
SHOW_CONFIG_INFO	Y
EDITOR	
REPORTER	

This UT_CONFIG record is created when the "`utconfig.autocompile`" procedure is run.

UT_PACKAGE Table

ID	1
SUITE_ID	
OWNER	UTP
NAME	BETWNSTR
DESCRIPTION	
SAMEPACKAGE	N
PREFIX	ut_
DIR	
SEQ	1
EXECUTIONS	2
FAILURES	0
LAST_STATUS	SUCCESS
LAST_START	06-JAN-2016 07:18:23
LAST_END	06-JAN-2016 07:18:24
LAST_RUN_ID	2

After the "BETWNSTR" unit test example is run, only 4 tables are populated with data. This UT_PACKAGE record is created the first time the "utPLSQL.test ('betwnstr')" procedure is run.

UT_UTP Table

ID	1
DESCRIPTION	
PREFIX	
PROGRAM	BETWNSTR
OWNER	UTP
FILENAME	
FREQUENCY	
PROGRAM_MAP	
DECLARATIONS	
SETUP	
TEARDOWN	
EXCEPTIONS	
PER_METHOD_SETUP	
NAME	
UTP_OWNER	
PROGRAM_FILENAME	
DIRECTORY	
PROGRAM_DIRECTORY	

This UT_UTP record is created the first time the "utPLSQL.test ('betwnstr')" procedure is run. It records a subset of data already recorded in the UT_PACKAGE table.

UTR_UTP Table

RUN_ID	RUN_BY	PROFILER_RUN_ID	UTP_ID	START_ON	END_ON	STATUS
1	(null)	(null)	1	05-JAN-2016 18:06:08	05-JAN-2016 18:06:08	SUCCESS
2	(null)	(null)	1	06-JAN-2016 07:18:23	06-JAN-2016 07:18:24	SUCCESS

This is the UTR_UTP table after 2 runs of the "BETWNSTR" unit test demo.

UTR_OUTCOME Table

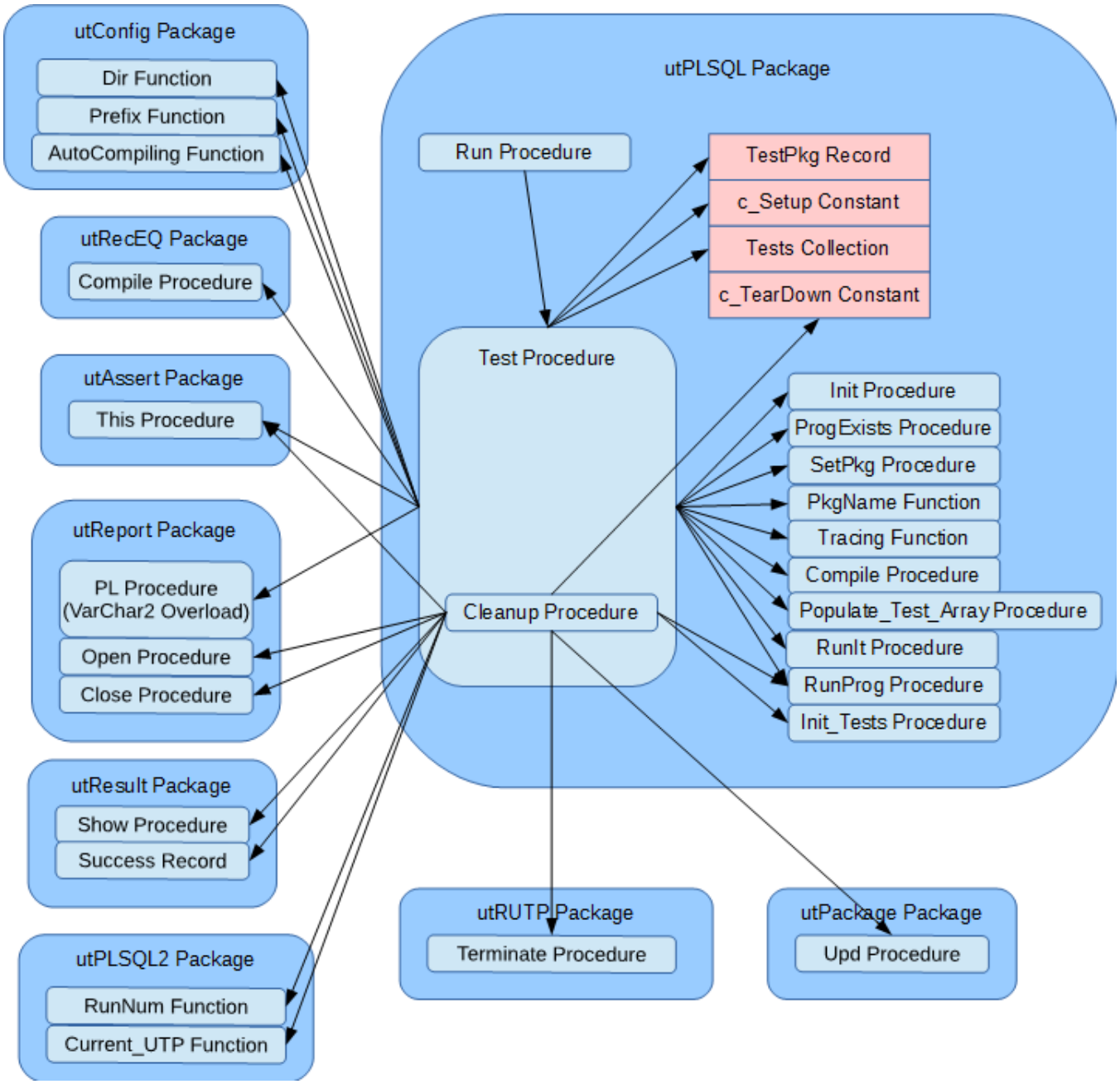
RUN_ID	OUTCOME_ID	START_ON	END_ON	STATUS	DESCRIPTION	TC_RUN_ID
1	-1	(null)	05-JAN-2016 18:06:08	SUCCESS	betwnstr.UT_BETWNSTR: EQ "Typical valid usage" Expected "cde" and ...	1
1	-2	(null)	05-JAN-2016 18:06:08	SUCCESS	betwnstr.UT_BETWNSTR: ISNULL "NULL start" Expected "" and got ""	2
1	-3	(null)	05-JAN-2016 18:06:08	SUCCESS	betwnstr.UT_BETWNSTR: ISNULL "NULL end" Expected "" and got ""	3
1	-4	(null)	05-JAN-2016 18:06:08	SUCCESS	betwnstr.UT_BETWNSTR: ISNULL "End smaller than start" Expected "" ...	4
1	-5	(null)	05-JAN-2016 18:06:08	SUCCESS	betwnstr.UT_BETWNSTR: EQ "End larger than string length" Expected ...	5
2	-1	(null)	06-JAN-2016 07:18:24	SUCCESS	betwnstr.UT_BETWNSTR: EQ "Typical valid usage" Expected "cde" and ...	1
2	-2	(null)	06-JAN-2016 07:18:24	SUCCESS	betwnstr.UT_BETWNSTR: ISNULL "NULL start" Expected "" and got ""	2
2	-3	(null)	06-JAN-2016 07:18:24	SUCCESS	betwnstr.UT_BETWNSTR: ISNULL "NULL end" Expected "" and got ""	3
2	-4	(null)	06-JAN-2016 07:18:24	SUCCESS	betwnstr.UT_BETWNSTR: ISNULL "End smaller than start" Expected "" ...	4
2	-5	(null)	06-JAN-2016 07:18:24	SUCCESS	betwnstr.UT_BETWNSTR: EQ "End larger than string length" Expected ...	5

This is the UTR_OUTCOME table after 2 runs of the "BETWNSTR" unit test demo. Of particular interest is the data in the DESCRIPTION field. Test failures are also recorded the same way. This column is a truncation of 6 data elements:

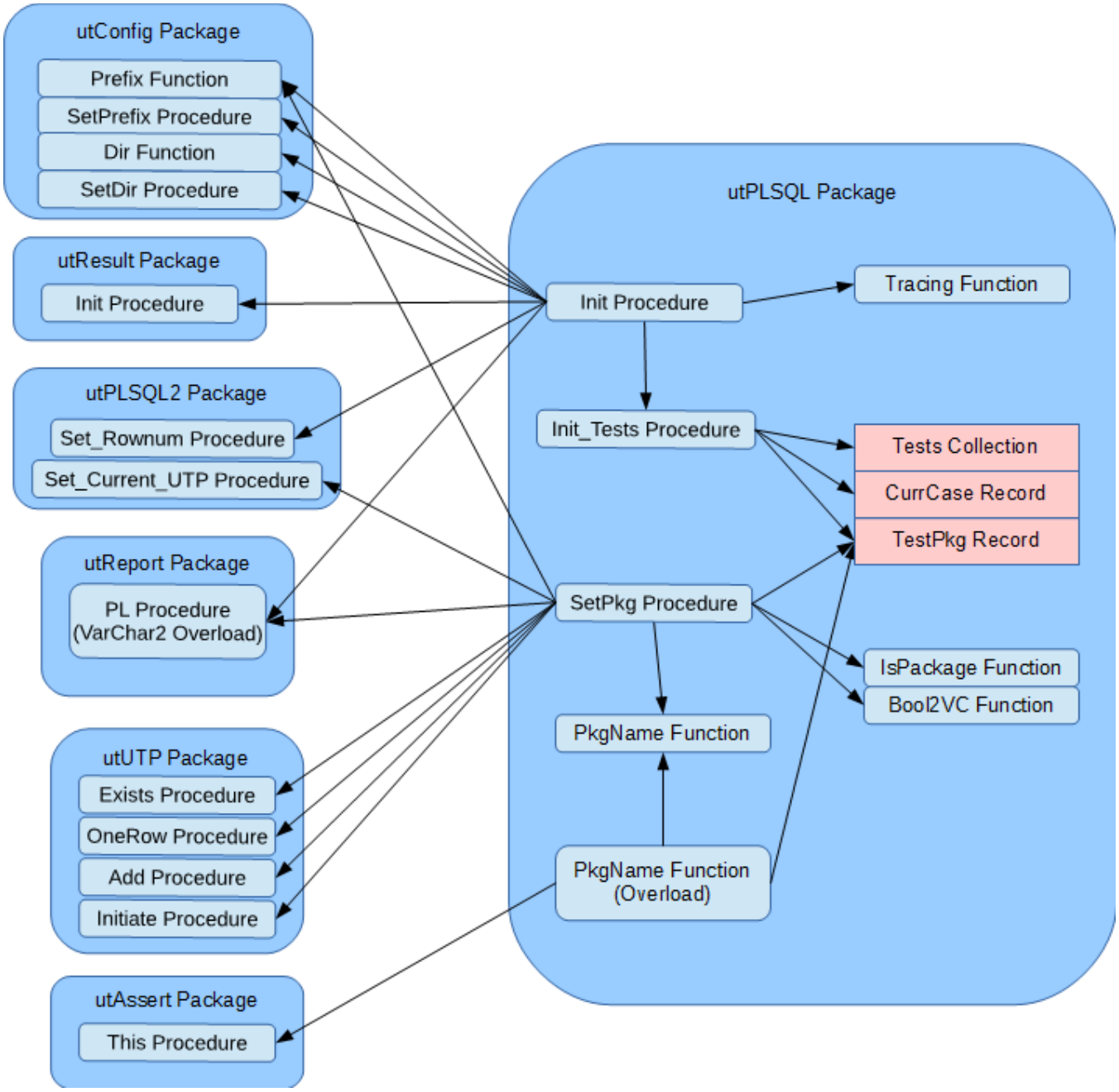
- Package Name
- Procedure Name from the Unit Test Package
- Assertion Test Name
- Assertion Test Message
- Expected Results
- Actual Results

Appendix G - Legacy utPLSQL Selected Call Trees

UTPLSQL RUN and TEST Procedures

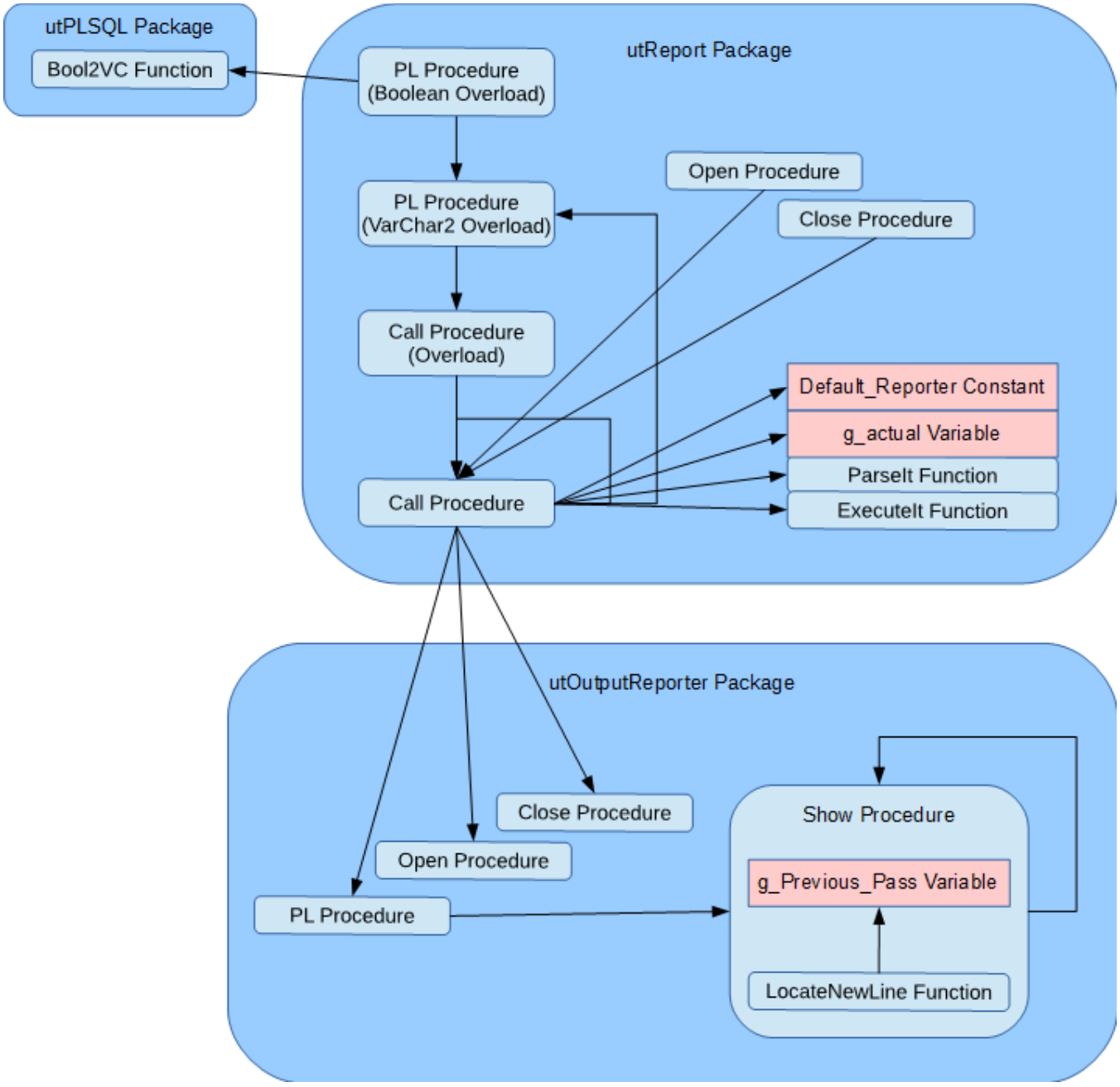


UTPLSQL Initialization



UTPLSQL Main Execution

utReport and utOutputReport Packages



utResult Show & utResult2 Report Procedures

From utAssert to utOutputReporter

(Some calls within the utReport Package have been omitted for clarity.)