**CREATE** **OR** **REPLACE** **PACKAGE** env

 **AUTHID** **CURRENT\_USER**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*%author*

*Bill Coulam (bcoulam@dbartisans.com)*

 *Package of routines that get or set information about the user's session and*

 *execution environment.*

*%warn*

 *I worry that wrapping calls to SYS\_CONTEXT will bypass Oracle optimizations*

 *for SYS\_CONTEXT, optimizations that allow SYS\_CONTEXT to sail past the usual*

 *limitations of calling PL/SQL functions from within SQL statements. I did a*

 *brief test on a table with 500K rows. The cost reported by the CBO was greater,*

 *but the response time was about the same. If you discover that calling these*

 *wrapped versions of SYS\_CONTEXT calls is slower, by all means, use the direct*

 *calls to SYS\_CONTEXT instead.*

*<pre>*

*Artisan Date Comments*

*============ ========= ========================================================*

*bcoulam 1997Dec30 Creation*

*bcoulam 2008Jan20 Added a few more functions.*

*bcoulam 2008May19 Added context functions, caller\_meta, get\_routine\_nm,*

 *line\_num\_here, and fixed private bundle\_stack\_lines to*

 *work with both 10g and 9i call stacks. Eliminated*

 *redundant get\_app\_id and get\_app\_cd functions. Renamed*

 *get\_current\_user to get\_current\_schema (the current\_user*

 *USERENV attribute is deprecated as of 10g). Simplified*

 *get\_env\_nm. Also added set and clear context routines.*

*bcoulam 2008Aug18 Added vld\_path\_format to ensure directory paths end in a*

 *slash.*

*<i>*

 *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ LGPL License \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

 *Copyright (C) 1997-2008 Bill Coulam*

 *This library is free software; you can redistribute it and/or*

 *modify it under the terms of the GNU Lesser General Public*

 *License as published by the Free Software Foundation; either*

 *version 2.1 of the License, or (at your option) any later version.*

 *This library is distributed in the hope that it will be useful,*

 *but WITHOUT ANY WARRANTY; without even the implied warranty of*

 *MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU*

 *Lesser General Public License for more details.*

 *You should have received a copy of the GNU Lesser General Public*

 *License along with this library; if not, write to the Free Software*

 *Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

**AS**

*--------------------------------------------------------------------------------*

*-- PUBLIC CURSORS*

*--------------------------------------------------------------------------------*

*--------------------------------------------------------------------------------*

*-- PUBLIC TYPES*

*--------------------------------------------------------------------------------*

*-- basic trace structure type*

**TYPE** trace\_info **IS** **RECORD** (

 **module** v$session.module%**TYPE**,

 action v$session.action%**TYPE**,

 client\_info v$session.client\_info%**TYPE**

);

*-- trace stack datatype*

**TYPE** tar\_trace\_info **IS** **TABLE** **OF** trace\_info **INDEX** **BY** **PLS\_INTEGER**;

*--------------------------------------------------------------------------------*

*-- PUBLIC CONSTANTS, VARIABLES, EXCEPTIONS, ETC.*

*--------------------------------------------------------------------------------*

DOMAIN **CONSTANT** **VARCHAR2**(30) := 'ldschurch.org';

APP\_CORE\_CTX **CONSTANT** **VARCHAR2**(20) := 'app\_core\_ctx';

*-- Global in-memory structure to keep track of the trace info used by the tag/untag*

*-- routines. This is used to reset the trace info back to its previous values when the*

*-- calling module finishes and returns control to the calling routine.*

g\_trace\_stack tar\_trace\_info;

*--------------------------------------------------------------------------------*

*-- PUBLIC FUNCTIONS*

*--------------------------------------------------------------------------------*

*/\*\*-----------------------------------------------------------------------------*

*Simple wrappers around SYS\_CONTEXT functions and environment metadata readily*

*available in the data dictionary.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** get\_client\_id **RETURN** **VARCHAR2**;

**FUNCTION** get\_client\_ip **RETURN** **VARCHAR2**;

**FUNCTION** get\_client\_host **RETURN** **VARCHAR2**;

**FUNCTION** get\_client\_os\_user **RETURN** **VARCHAR2**;

**FUNCTION** get\_client\_program **RETURN** **VARCHAR2**; *-- derived from [g]v$session. Not in USERENV*

**FUNCTION** get\_client\_module **RETURN** **VARCHAR2**;

**FUNCTION** get\_client\_action **RETURN** **VARCHAR2**;

**FUNCTION** get\_session\_user **RETURN** **VARCHAR2**;

**FUNCTION** get\_current\_schema **RETURN** **VARCHAR2**; *-- could be different from session user if current\_schema was altered*

**FUNCTION** get\_db\_version **RETURN** **NUMBER**; *-- stand-in for older systems missing DBMS\_DB\_VERSION*

**FUNCTION** get\_db\_name **RETURN** **VARCHAR2**;

**FUNCTION** get\_db\_instance\_name **RETURN** **VARCHAR2**; *-- if RAC db, returns SID/SERVICE NAME otherwise*

**FUNCTION** get\_db\_instance\_id **RETURN** **NUMBER**; *-- if RAC db, returns 1 otherwise*

**FUNCTION** get\_server\_host **RETURN** **VARCHAR2**; *-- name of db host*

*-- Note: get\_sid and get\_session\_id do the same thing; get\_sid is just easier to remember.*

**FUNCTION** get\_sid **RETURN** **INTEGER**; *-- db session ID ([g]v$session.sid)*

**FUNCTION** get\_session\_id **RETURN** **INTEGER**; *-- db session ID ([g]v$session.sid)*

**FUNCTION** get\_os\_pid **RETURN** **INTEGER**; *-- db host operating system process ID*

*--FUNCTION get\_global\_context\_memory RETURN VARCHAR2;*

**FUNCTION** get\_schema\_email\_address **RETURN** **VARCHAR2**; *--fake email address db.schema@host*

*/\*\*-----------------------------------------------------------------------------*

*get\_db\_id:*

 *Uses the db\_name returned by sys\_context to query APP\_DB and return a numeric*

 *ID for the current database.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** get\_db\_id **RETURN** **INTEGER**;

*/\*\*-----------------------------------------------------------------------------*

*get\_db\_alias*

 *Given a database identifier, will return the database alias. If the identifier*

 *is not given, will return the alias for the local database.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** get\_db\_alias(i\_db **IN** **VARCHAR2** **DEFAULT** **NULL**) **RETURN** **VARCHAR2**;

*/\*\*-----------------------------------------------------------------------------*

*get\_app\_id:*

 *If the app\_cd is given, returns the app\_id from the APP table.*

 *If the app\_cd is missing, will return app\_id from the in-memory record private*

 *to ENV. If this has not been filled yet, it will attempt to determine the*

 *application transparently using values from the USERENV context and the DB Name,*

 *matching up to data in APP, APP\_DB and APP\_ENV. In environments where multiple*

 *applications share the same owning schema, they must set the app\_cd into the*

 *default env.APP\_CORE\_CTX context at the time of connection. They can do this by*

 *ensuring they call env.init\_client\_ctx and passing in the app\_cd at the start of*

 *their session. Or they can do this by calling env.set\_ctx\_val directly.*

 *If the app\_cd cannot be determined, an error is raised.*

*%param i\_app\_cd Name of the application, as stored in table APP.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** get\_app\_id (i\_app\_cd **IN** app.app\_cd%**TYPE** **DEFAULT** **NULL**) **RETURN** **NUMBER**;

*/\*\*-----------------------------------------------------------------------------*

*get\_app\_cd:*

 *If the app\_id is given, returns the app code from the APP table.*

 *If the app\_id is missing, will transparently attempt to get the app\_id using*

 *get\_app\_id().*

 *If the app\_cd cannot be determined, will return NULL.*

*%param i\_app\_id ID of the application, as stored in table APP.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** get\_app\_cd(i\_app\_id **IN** app.app\_id%**TYPE** **DEFAULT** **NULL**) **RETURN** **VARCHAR2**;

*/\*\*-----------------------------------------------------------------------------*

*get\_env\_nm:*

 *Queries APP\_ENV to determine the name of the current environment, given the*

 *current schema, database name and application.*

*%param i\_app\_cd Optional application code.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** get\_env\_nm(i\_app\_cd **IN** app.app\_cd%**TYPE** **DEFAULT** **NULL**) **RETURN** **VARCHAR2**;

*/\*\*-----------------------------------------------------------------------------*

*get\_dir\_path:*

 *Queries all\_directories to find the directory path behind a given 9i-style*

 *directory name.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** get\_dir\_path(i\_dir\_nm **IN** **VARCHAR2**) **RETURN** **VARCHAR2**;

*/\*\*-----------------------------------------------------------------------------*

*vld\_path\_format:*

 *Ensures that there is a directory slash character after a given path. If the*

 *given path does not end in a slash, one will be appended. This is useful for*

 *routines that piece together full paths, where the content and validity of the*

 *path piece is uncertain. When Oracle directory objects are created, they may or*

 *may not have the trailing slash.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** vld\_path\_format (i\_path **IN** **VARCHAR2**) **RETURN** **VARCHAR2**;

*/\*\*-----------------------------------------------------------------------------*

*who\_called\_me:*

 *By default, returns the name of the package or standalone one level further up*

 *in the call stack, which represents the caller of the routine that called this*

 *function. When called indirectly by another layer in the framework, the stack*

 *level needs to be increased from the default to find out who the real*

 *caller's caller is.*

*%credit*

 *Inspiration for this routine name and the code behind it came from Tom Kyte.*

*%param i\_stack\_level The depth in the stack to look for caller info.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** who\_called\_me(i\_stack\_level **IN** **PLS\_INTEGER** **DEFAULT** 2) **RETURN** **VARCHAR2**;

*/\*\*-----------------------------------------------------------------------------*

*who\_am\_i:*

 *By default, returns the name of the package or standalone that called this*

 *function. When called indirectly by another layer in the framework, the stack*

 *level needs to be increased from the default to find out who the real caller is.*

*%credit*

 *Inspiration for this routine name and the code behind it came from Tom Kyte.*

*%param i\_stack\_level The depth in the stack to look for caller info.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** who\_am\_i(i\_stack\_level **IN** **PLS\_INTEGER** **DEFAULT** 1) **RETURN** **VARCHAR2**;

*/\*\*-----------------------------------------------------------------------------*

*get\_routine\_nm:*

 *Given a package name and line number, returns the name of the routine within*

 *which that line number currently falls. The ability of a subroutine to*

 *introspect and find its own name is a basic ability of most programming and*

 *scripting languages, but not PL/SQL. Hence the reason for this routine.*

 *By parsing the call stack (available in most versions of Oracle) or using the*

 *new $$PLSQL\_UNIT and $$PLSQL\_LINE directives, we can get at the package and*

 *line. Either of these methods can produce the inputs required for this*

 *function.*

*%note*

 *This was originally a private function, meant to be used exclusively by the*

 *caller\_meta() routine. However, since it could be useful in other contexts, it*

 *was exposed. The logging library indirectly uses this routine heavily so that*

 *the callers of the logging routines do not have to pass in their location, name*

 *or containing package explicitly. If you only desire to use this function for*

 *logging purposes, use the LOGS package instead, and inherit this ability by*

 *default.*

*%caveat*

 *If you still need to call this function, know that it is only useful for*

 *\_packaged\_ routines.*

 *The accuracy of this function depends on the code following one simple*

 *convention, which is:*

 *Always immediately follow the PROCEDURE and FUNCTION declaration with its name.*

 *Example: CREATE OR REPLACE PACKAGE test AS*

 *...*

 *PROCEDURE first\_proc( <-- fine*

 *...*

 *);*

 *FUNCTION get\_val... <-- fine*

 *PROCEDURE -- inline comment about this proc*

 *second\_proc; <-- will not work with env.get\_routine\_nm()*

 *END test;*

 *If you allow the routine name to go on a line separate from its declaration,*

 *this function will not work and return NULL.*

*%design*

 *Two designs were tested, one where the metadata for each package was stored*

 *in a table and maintained by trigger (nature of DDL in AFTER ALTER triggers*

 *requires that the trigger submit jobs to update the table), and one where the*

 *metadata is gathered at the time of request. Running a test of 1700 random*

 *line numbers within a schema with 140 packages, the persistent table version*

 *returned accurate routine names in 1.3 seconds for 1700 total calls, whereas*

 *the dynamic version got the same results in 2.8 seconds. Since get\_routine\_nm*

 *wouldn't be called that heavily within most production environments, I decided*

 *to eliminate the overhead of the extra moving parts and stick with the dynamic*

 *version instead. That is why this routine just uses one complex SQL statement,*

 *instead of a static table or materialized result set.*

*%param i\_pkg\_nm Name of the package in which to find the routine name by line*

 *number.*

*%param i\_line\_num The line number which the caller is claiming as the location*

 *of the introspection request.*

*------------------------------------------------------------------------------\*/*

**FUNCTION** get\_routine\_nm

(

 i\_pkg\_nm **IN** **VARCHAR2**,

 i\_line\_num **IN** **INTEGER**

) **RETURN** **VARCHAR2**;

*/\*\*-----------------------------------------------------------------------------*

*line\_num\_here:*

 *Looks in the call stack to the given depth and returns the line number from*

 *which line\_num\_here was called.*

*%note*

 *This is not necessary from 9.2.0.6 onwards where you can use $$PLSQL\_LINE to*

 *get the same result. However, in 9.2.0.6, you have to set the*

 *\_plsql\_conditional\_compilation flag to TRUE in order to use it.*

*%usage*

 *<code>*

 *-- When you want to record a line number other than the line at which the*

 *-- LOGS routine is called. This is analagous to the old method of setting*

 *-- a "marker" variable before each chunk of code, which would get recorded*

 *-- with any error/info logging for context and later research:*

 *--l\_line := $$PLSQL\_LINE; -- implemenation for >= 9.2.0.6*

 *l\_line := env.line\_num\_here; -- implementation for <= 9.2.0.5*

 *mypkg.do\_something\_useful(i\_date, l\_length);*

 *logs.dbg(i\_msg=>'Did something useful', i\_line\_num => l\_line);*

 *-- When you are OK with recording the line number as the line from which the*

 *-- call to logs was made, use this simpler method instead:*

 *-- >= 9.2.0.6*

 *logs.info('Awaiting pipe message', 'DBMS\_PIPE Listener', $$PLSQL\_LINE);*

 *-- <= 9.2.0.5*

 *logs.info('Awaiting pipe message', 'DBMS\_PIPE Listener', env.line\_num\_here);*

 *-- Remember that the LOGS routines transparently determine routine name and*

 *-- line number for you. So unless you want to use a custom name, like*

 *-- DBMS\_PIPE Listener in the examples above, don't worry about line number,*

 *-- for example:*

 *logs.info('Awaiting pipe message');*

*------------------------------------------------------------------------------\*/*

**FUNCTION** line\_num\_here(i\_stack\_level **IN** **PLS\_INTEGER** **DEFAULT** 1) **RETURN** **INTEGER**;

*--------------------------------------------------------------------------------*

*-- PUBLIC PROCEDURES*

*--------------------------------------------------------------------------------*

*/\*\*-----------------------------------------------------------------------------*

*caller\_meta:*

 *Returns all the metadata about the caller at the given level in the call stack.*

 *It is anticipated that the only consumer of this proc will be the LOGS library*

 *routines (msg, err, warn, info and dbg).*

 *The caller data includes the fully qualified routine name, which is the name of*

 *the type body, function, proc, trigger or package.routine\_name. If the caller*

 *is an anonymous block, the name will be ANONYMOUSBLOCK.*

*%param o\_owner The schema name of the owner of the caller DB object.*

*%param o\_caller\_type The PL/SQL unit type, including ANONYMOUS BLOCK*

*%param o\_unit\_nm The PL/SQL unit name. For packages, this is limited to the package*

 *name.*

*%param o\_routine\_nm For packages, this is the the full name of the calling object.*

 *Will be package.routine\_nm for packaged routines.*

*%param o\_line\_num The line number in the call stack from which the call was made.*

*%param i\_stack\_level How deep to look in the stack for for the callers metadata.*

 *Defaults to 1 level deep (immediate caller). The framework*

 *components must use stack level 2 to get one layer above*

 *where they sit in the call stack.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** caller\_meta

(

 o\_owner **OUT** typ.t\_maxobjnm,

 o\_caller\_type **OUT** user\_objects.object\_type%**TYPE**,

 o\_unit\_nm **OUT** user\_objects.object\_name%**TYPE**,

 o\_routine\_nm **OUT** app\_log.routine\_nm%**TYPE**,

 o\_line\_num **OUT** app\_log.line\_num%**TYPE**,

 i\_stack\_level **IN** **PLS\_INTEGER** **DEFAULT** 1

);

*/\*\*-----------------------------------------------------------------------------*

*tag\_session/tag:*

 *Sets MODULE, ACTION and CLIENT\_INFO in v$session to the provided values. Use*

 *this routine frequently to instrument your code and DDL/DML upgrade scripts.*

 *Since this places custom tags in v$session, it gives DBAs more visibility into*

 *the systems they manage, and a better ability to measure, tune, find and track.*

 *Remember to clear these values out using env.untag\_session, or else the values*

 *will remain for the duration of the session, possibly causing those*

 *investigating issues to pursue the wrong path.*

*%usage*

 *<code>*

 *exec env.tag\_session('CR53885','Re-create Constraint','CONTACTS\_UK');*

 *ALTER TABLE contacts*

 *DROP CONSTRAINT contacts\_uk*

 *...*

 *ALTER TABLE contacts*

 *ADD CONSTRAINT contacts\_uk*

 *...*

 *exec ddl\_utils.analyze\_index('CONTACTS\_UK');*

 *exec env.untag\_session;*

 *</code>*

*%param i\_module The governing "module", usually the change request/ticket#.*

 *The PL/SQL package name is also a frequently-used value.*

 *If not given, the name of the calling package (if any) will be*

 *determined transparently.*

 *Limited to 48 characters.*

*%param i\_action The current "action", usually something like "Create Index",*

 *"Move Table", etc. The packaged procedure/function name is also a*

 *frequently-used value. If not given, the name of the calling*

 *routine will be determined transparently.*

 *Limited to 32 characters.*

*%param i\_info The detail of the current step, usually the name of the table,*

 *index or constraint being created/altered/queried. If not supplied,*

 *the line number from which tag\_session was called in the calling*

 *routine will be used instead.*

 *Limited to 64 characters.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** tag\_session

(

 i\_module **IN** **VARCHAR2**,

 i\_action **IN** **VARCHAR2**,

 i\_info **IN** **VARCHAR2**

);

*-- New version that doesn't require caller to pass anything in. Also shorter name.*

**PROCEDURE** tag

(

 i\_module **IN** **VARCHAR2** **DEFAULT** **NULL**,

 i\_action **IN** **VARCHAR2** **DEFAULT** **NULL**,

 i\_info **IN** **VARCHAR2** **DEFAULT** **NULL**

);

*/\*\*-----------------------------------------------------------------------------*

*untag\_session/untag:*

 *Sets MODULE, ACTION and CLIENT\_INFO in v$session to NULL.*

*%note*

 *Make sure to call this after calling tag\_session(). Otherwise the info fed to*

 *tag\_session will remain attached to your session, fooling administrators into*

 *thinking your session is still working on the module indicated in v$session,*

 *when in fact your session has ended or moved on to other actions.*

*%param i\_restore\_prior\_tag If TRUE (the default), will attempt to read the global*

 *variable, gv\_trace\_info to see if prior trace data is*

 *stored there. If so, it will use these values instead*

 *of NULL. This allows modules called by other modules*

 *to restore the module/action/info as it was prior to*

 *the current module call and use of tag/tag\_session.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** untag\_session(i\_restore\_prior\_tag **IN** **BOOLEAN** **DEFAULT** **TRUE**);

*-- New version that doesn't require caller to pass anything in. Also shorter name.*

**PROCEDURE** untag(i\_restore\_prior\_tag **IN** **BOOLEAN** **DEFAULT** **TRUE**);

*/\*\*-----------------------------------------------------------------------------*

*set\_ctx\_val:*

 *You associate this routine with an application-specific context during*

 *application context creation. Then call this routine when setting the values*

 *of attributes within the context. If the application context (the Oracle docs*

 *seem to use "application context" and context namespace interchangeably) is*

 *not named, the default namespace for the Core framework will be used.*

*%note*

 *The Core framework uses this routine to maintain the value of the in-memory*

 *debug flag parameter.*

*%usage*

 *<code>*

 *CREATE CONTEXT my\_ctx USING env.set\_ctx\_val;*

 *exec env.set\_ctx\_val('remote\_login\_attempts','5','my\_ctx');*

 *-- followed by calls to PL/SQL stored objects that include controls on*

 *-- remote login attempts by querying the in-memory context value using*

 *-- SYS\_CONTEXT('my\_ctx','remote\_login\_attempts')*

 *</code>*

*%param i\_attr\_nm Name of the attribute whose value will be set in the context*

 *for the current session.*

*%param i\_attr\_val Value of the attribute within the application context.*

*%param i\_ctx\_nm Name of the application context. Defaults to framework's context.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** set\_ctx\_val

(

 i\_attr\_nm **IN** **VARCHAR2**,

 i\_attr\_val **IN** **VARCHAR2**,

 i\_ctx\_nm **IN** **VARCHAR2** **DEFAULT** APP\_CORE\_CTX

);

*/\*\*-----------------------------------------------------------------------------*

*set\_app\_cd:*

 *Takes a short code for a given application (found listed in the APP table),*

 *and sets it as the value for the "app\_cd" attribute in the Core application*

 *context.*

 *This routine is meant to be called by jobs, or DBAs manually running routines*

 *by anonymous block. In these situations the client or user ID might not be*

 *readily known or relevant, and only the app\_cd is needed to activate the*

 *framework's dynamic views. This could be called at the top of the "what" block*

 *in the job. A great place for the call is within the initialization section of*

 *a package. If you are designing a system that resides in its own dedicated*

 *schema, this routine need only be called once from an AFTER LOGON trigger. But*

 *if you have the unfortunate luck of working in a schema that serves multiple*

 *applications, this routine will need to be called explicitly by the packages*

 *and jobs dedicated to a given system.*

*%param i\_app\_cd Application code as found in APP.APP\_CD*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** set\_app\_cd

(

 i\_app\_cd **IN** app.app\_cd%**TYPE**

);

*/\*\*-----------------------------------------------------------------------------*

*set\_current\_schema:*

 *Takes an Oracle account/schema name, and places it in the session's client*

 *context memory structure for later use when read by get\_current\_schema.*

 *For most systems where the application resides in one schema, and calls upon*

 *services in the framework in the same or another schema, this routine is not*

 *necessary as the framework will determine the current schema being used for*

 *execution dynamically. I call this a two layer use of the framework.*

 *However, for systems where schema A calls upon a definer-rights stored*

 *routine in schema B, or a view or trigger in schema B (neither of which can*

 *use invoker-rights), which make use of the framework, any attempt to get the*

 *application's current schema will return schema B, not schema A as it should.*

 *I call this a three or N-layer use of the framework, which it really wasn't*

 *designed for.*

 *So this routine was added to allow a session to statically place its object-*

 *owning or DB access account name (see app\_env.owner\_account and access\_account*

 *columns and how they help map applications and databases to named environments)*

 *into memory where it can be retrieved by the framework calls within schema B.*

 *This call would ideally be done within an AFTER LOGON trigger in the schema to*

 *which the application server connects.*

*%param i\_schema\_nm Valid Oracle account name. Will be uppercased before being*

 *stored. So ensure it matches a valid account value in*

 *APP\_ENV.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** set\_current\_schema

(

 i\_schema\_nm **IN** app\_env.access\_account%**TYPE**

);

*/\*\*-----------------------------------------------------------------------------*

*init\_client\_ctx:*

 *Takes a user identifier from the caller (usually the presentation layer that*

 *served up the login screen to the user), and places it in the session's*

 *client\_identifier USERENV application context area. This can be used by*

 *standard Oracle auditing, or FGA, or custom trigger-based auditing, to report*

 *who (within the application's users sharing the connection pool) did what, and*

 *when.*

*%param i\_client\_id User's login ID, be it employee ID, name, LDAP DN, whatever*

 *can identify the end user. If this is an automated process,*

 *assign it a name and use that name consistently here and when*

 *logging.*

*%param i\_client\_ip Optional IP address. If using connection pools and shared schemas,*

 *this will default to the address of the application server,*

 *unless you pass it the user's IP address explicitly.*

*%param i\_client\_host Optional name of the machine or terminal from the which the*

 *user is logging in to use the application. Again, for*

 *applications using connection pools, this will default to*

 *the application server's host name unless the front end*

 *passes the client machine name explicitly.*

*%param i\_client\_os\_user Optional operating system login ID. This is more useful*

 *when the user is connected directly to the database,*

 *or for automated processes. But if the front end layer*

 *has this information, they should feel free to pass it in.*

*%param i\_app\_cd An application code from APP.APP\_CD. Mandatory if this is an*

 *environment where multiple applications share a single schema.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** init\_client\_ctx

(

 i\_client\_id **IN** **VARCHAR2**,

 i\_client\_ip **IN** **VARCHAR2** **DEFAULT** **NULL**,

 i\_client\_host **IN** **VARCHAR2** **DEFAULT** **NULL**,

 i\_client\_os\_user **IN** **VARCHAR2** **DEFAULT** **NULL**,

 i\_app\_cd **IN** **VARCHAR2** **DEFAULT** **NULL**

);

*/\*\*-----------------------------------------------------------------------------*

*reset\_client\_ctx:*

 *Must be called by the front-end layer controlling transactions and access to the*

 *connection pool. This empties the client context and resets package state, so that*

 *the next user who inherits these in-memory objects doesn't also inherit the same*

 *values.*

*%warn*

 *This routine re-initializes several things, including the session state that*

 *held the dbms\_output buffer. So make sure you are pulling relevant text out of the*

 *buffer (if you are using dbms\_output.get\_line(s)) before you call reset\_client\_ctx.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** reset\_client\_ctx;

*/\*\*-----------------------------------------------------------------------------*

*clear\_ctx\_val:*

 *This routine will set the value of the attribute within the given namespace to*

 *NULL.*

*%usage*

 *Call this routine when the value of an in-memory application context attribute*

 *is no longer needed. This prevents the value from being inherited by other*

 *connections or sessions (I lose track of which is which).*

*%param i\_attr\_nm Name of the attribute whose value will be set to NULL.*

*%param i\_ctx\_nm Name of the application context. Defaults to framework's context.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** clear\_ctx\_val

(

 i\_attr\_nm **IN** **VARCHAR2**,

 i\_ctx\_nm **IN** **VARCHAR2** **DEFAULT** APP\_CORE\_CTX

);

*/\*\*-----------------------------------------------------------------------------*

*clear\_ctx:*

 *This routine will set the value of all attributes within the given namespace to*

 *NULL.*

*%usage*

 *Call this routine when a session is over and the connection is about to be*

 *returned to the pool. This prevents the values in the namespace from being*

 *inherited by other connections or sessions (I lose track of which is which).*

*%param i\_ctx\_nm Name of the application context. Defaults to framework's context.*

*------------------------------------------------------------------------------\*/*

**PROCEDURE** clear\_ctx

(

 i\_ctx\_nm **IN** **VARCHAR2** **DEFAULT** APP\_CORE\_CTX

);

**END** env;