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

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

*%author*

*Bill Coulam (bcoulam@dbartisans.com)*

*<pre>*

*Artisan Date Comments*

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

*bcoulam 1997Dec30 Creation*

*<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**

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

*-- PACKAGE CONSTANTS, VARIABLES, TYPES, EXCEPTIONS*

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

*--gc\_pkg\_nm CONSTANT user\_source.name%TYPE := 'env';*

**TYPE** tr\_client\_ctx **IS** **RECORD**(

client\_id app\_log.client\_id%**TYPE**,

client\_ip app\_log.client\_ip%**TYPE**,

client\_host app\_log.client\_host%**TYPE**,

client\_os\_user app\_log.client\_os\_user%**TYPE**,

client\_program **VARCHAR2**(100),

client\_module **VARCHAR2**(100),

client\_action **VARCHAR2**(100),

session\_user **VARCHAR2**(30),

**current\_schema** **VARCHAR2**(30),

app\_id app.app\_id%**TYPE**

);

**TYPE** tr\_server\_ctx **IS** **RECORD** (

db\_version **INTEGER**,

db\_name **VARCHAR2**(30),

db\_instance\_name **VARCHAR2**(30),

server\_host **VARCHAR2**(40),

**sid** **INTEGER**,

os\_pid **INTEGER**

);

gr\_client\_ctx tr\_client\_ctx; *-- global record of user info*

gr\_server\_ctx tr\_server\_ctx; *-- global record of DB server info*

gr\_empty\_client\_ctx tr\_client\_ctx; *-- used for re-initializations of session context for a user*

**TYPE** tr\_stack\_data **IS** **RECORD** (

owner **VARCHAR2**(30)

, obj\_nm **VARCHAR2**(30)

, obj\_type **VARCHAR2**(30)

, line\_num **NUMBER**

);

**TYPE** tar\_stack\_data **IS** **TABLE** **OF** tr\_stack\_data **INDEX** **BY** **BINARY\_INTEGER**;

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

*-- PRIVATE FUNCTIONS AND PROCEDURES*

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

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

*ins*

*This private INSERT to app\_log routine is handy for debugging the ENV package.*

*ENV is supposed to be the lowest level in the package hierarchy (along with CNST,*

*TYP, DT, STR and NUM), so it cannot call routines in the LOGS or APP\_LOG\_API*

*packages.*

*Uncomment this routine and recompile if you need to debug something within ENV.*

*%design*

*Must use autonomous transaction in order to see the results immediately in the*

*table.*

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

**PROCEDURE** ins(i\_log\_txt **IN** **VARCHAR2**) **IS**

**PRAGMA** **AUTONOMOUS\_TRANSACTION**;

lr\_app\_log app\_log%**ROWTYPE**;

**BEGIN**

**SELECT** app\_log\_seq.NEXTVAL

**INTO** lr\_app\_log.log\_id

**FROM** dual;

lr\_app\_log.app\_id := 0;

lr\_app\_log.log\_ts := dt.get\_systs;

lr\_app\_log.sev\_cd := cnst.INFO;

lr\_app\_log.msg\_cd := msgs.DEFAULT\_MSG\_CD;

lr\_app\_log.routine\_nm := cnst.UNKNOWN\_STR;

lr\_app\_log.line\_num := **NULL**;

lr\_app\_log.log\_txt := i\_log\_txt;

lr\_app\_log.client\_id := get\_client\_id;

**INSERT** **INTO** app\_log

**VALUES** lr\_app\_log;

**COMMIT**; *-- must be here for autonomous to work*

**END** ins;

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

*bundle\_stack\_lines:*

*Gets the current call\_stack and parses useful information out of it into*

*an array for further processing as the caller sees fit.*

*In 9i a call stack looks like this:*

*----- PL/SQL Call Stack -----*

*object line object*

*handle number name*

*0x75c30c5c 94 package body APP\_CORE.ENV*

*0x75c30c5c 569 package body APP\_CORE.ENV*

*0x730f6920 58 package body APP\_CORE.BOTTOM*

*0x74be5fd0 13 package body APP\_CORE.MIDDLE*

*0x737e7d04 5 package body APP\_CORE.TOP*

*0x6e8d32e0 2 anonymous block*

*In 10g a call stack looks like this:*

*----- PL/SQL Call Stack -----*

*object line object*

*handle number name*

*2E943CFC 94 package body APP\_CORE.ENV*

*2E943CFC 569 package body APP\_CORE.ENV*

*2E908B0C 58 package body APP\_CORE.BOTTOM*

*2E909084 13 package body APP\_CORE.MIDDLE*

*2E9095FC 5 package body APP\_CORE.TOP*

*2E8AF158 2 anonymous block*

*lines 1-3 are header info we don't need*

*line 4 is ME, the immediate block where format\_call\_stack was just called*

*line 5 is MY Caller*

*line 6 is Their Caller*

*and so on...*

*Much of this is from Tom Kyte's "who\_called\_me" procedure.*

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

**PROCEDURE** bundle\_stack\_lines

(oar\_stack\_data **OUT** tar\_stack\_data)

**AS**

l\_call\_stack **VARCHAR2**(4096) **DEFAULT** DBMS\_UTILITY.format\_call\_stack;

l\_pos **PLS\_INTEGER** := 0;

l\_line **VARCHAR2**(255);

l\_cnt **PLS\_INTEGER** := 0;

lar\_stack\_data tar\_stack\_data;

**BEGIN**

*--dbms\_output.put\_line('bundle\_stack\_lines:');*

*--dbms\_output.put\_line(substr(l\_call\_stack,1,255));*

l\_pos := **INSTR**(l\_call\_stack,**CHR**(10),1,3); *--bypasses header lines*

l\_call\_stack := **SUBSTR**(l\_call\_stack, l\_pos + 1); *--removes header lines*

*-- loop through lines in stack and pull out good stuff into array*

**WHILE** (l\_call\_stack **IS** **NOT** **NULL**) **LOOP**

l\_cnt := l\_cnt + 1;

l\_pos := **INSTR**(l\_call\_stack,**CHR**(10)); *-- get pos of next linefeed*

**EXIT** **WHEN** (l\_pos **IS** **NULL** **OR** l\_pos = 0); *-- should exit when no more linefeeds*

l\_line := **SUBSTR**(l\_call\_stack, 1, l\_pos - 1);

l\_call\_stack := **SUBSTR**(l\_call\_stack, l\_pos + 1);

*-- Trip address off left side of line*

l\_line := **LTRIM**(**SUBSTR**(l\_line, **INSTR**(l\_line,**CHR**(32))));

*-- First thing we care about is the line number*

lar\_stack\_data(l\_cnt).line\_num := **TO\_NUMBER**(**SUBSTR**(l\_line, 1, **INSTR**(l\_line,**CHR**(32))-1));

*-- Need whatever's left after the line number*

l\_line := **LTRIM**(**SUBSTR**(l\_line, **LENGTH**(lar\_stack\_data(l\_cnt).line\_num)+2));

**IF** (l\_line **LIKE** 'pr%') **THEN**

l\_pos := **LENGTH**('procedure ');

**ELSIF** (l\_line **LIKE** 'fun%') **THEN**

l\_pos := **LENGTH**('function ');

**ELSIF** (l\_line **LIKE** 'package body%') **THEN**

l\_pos := **LENGTH**('package body ');

**ELSIF** (l\_line **LIKE** 'pack%') **THEN**

l\_pos := **LENGTH**('package ');

**ELSIF** (l\_line **LIKE** 'anonymous%') **THEN**

l\_pos := **LENGTH**('anonymous block ');

**ELSE**

l\_pos := **NULL**;

**END** **IF**;

**IF** (l\_pos **IS** **NOT** **NULL**) **THEN**

lar\_stack\_data(l\_cnt).obj\_type := **LTRIM**(**RTRIM**(**UPPER**(**SUBSTR**(l\_line, 1, l\_pos - 1))));

**ELSE**

lar\_stack\_data(l\_cnt).obj\_type := 'TRIGGER';

**END** **IF**;

l\_line := **SUBSTR**(l\_line, **NVL**(l\_pos, 1));

l\_pos := **INSTR**(l\_line, '.');

**IF** (l\_pos > 0) **THEN**

lar\_stack\_data(l\_cnt).owner := **LTRIM**(**RTRIM**(**SUBSTR**(l\_line, 1, l\_pos - 1)));

lar\_stack\_data(l\_cnt).obj\_nm := **LTRIM**(**RTRIM**(**SUBSTR**(l\_line, l\_pos + 1)));

**ELSE**

lar\_stack\_data(l\_cnt).owner := env.get\_current\_schema;

lar\_stack\_data(l\_cnt).obj\_nm := 'ANONYMOUSBLOCK';

**END** **IF**;

**END** **LOOP**;

**FOR** i **IN** lar\_stack\_data.FIRST..lar\_stack\_data.LAST **LOOP**

*-- these three lines for testing only*

*-- app\_log\_api.ins(i\_log\_txt => lar\_stack\_data(i).owner||'.'||lar\_stack\_data(i).obj\_type||'.'||*

*-- lar\_stack\_data(i).obj\_nm||' @ line '||lar\_stack\_data(i).line\_num,*

*-- i\_routine\_nm => 'env.bundle\_stack\_lines');*

**IF** (lar\_stack\_data(i).obj\_nm <> 'ENV') **THEN**

oar\_stack\_data(oar\_stack\_data.COUNT+1) := lar\_stack\_data(i);

**END** **IF**;

**END** **LOOP**;

**END** bundle\_stack\_lines;

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

*push\_trace\_stack*

*Private routine to place another set of tracing values onto the top of stack.*

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

**PROCEDURE** push\_trace\_stack

(

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

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

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

)

**IS**

l\_idx **INTEGER** := 0;

**BEGIN**

l\_idx := g\_trace\_stack.COUNT+1;

g\_trace\_stack(l\_idx).module := i\_module;

g\_trace\_stack(l\_idx).action := i\_action;

g\_trace\_stack(l\_idx).client\_info := i\_info;

**END** push\_trace\_stack;

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

*pop\_trace\_stack*

*Private routine to remove the last set of tracing values from the top of the*

*stack.*

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

**PROCEDURE** pop\_trace\_stack

**IS**

l\_idx **INTEGER** := 0;

**BEGIN**

l\_idx := g\_trace\_stack.COUNT;

**IF** (l\_idx = 0) **THEN**

**RETURN**;

**ELSE**

g\_trace\_stack.DELETE(l\_idx);

**END** **IF**;

**END** pop\_trace\_stack;

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

*-- PUBLIC FUNCTIONS AND PROCEDURES*

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

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

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

**BEGIN**

*-- This should have been set by init\_client\_ctx, when called by the front-end*

*-- at the start of the session or transaction. If it is not yet set, this*

*-- means the application is either not using this package, or someone logged*

*-- into an account with privs on the application schema, and is triggering*

*-- auditing code that requires this function.*

*--ins('get\_client\_id[gr\_client\_ctx.client\_id = '||gr\_client\_ctx.client\_id||']');*

**IF** (gr\_client\_ctx.client\_id **IS** **NULL**) **THEN**

gr\_client\_ctx.client\_id := **SYS\_CONTEXT**('userenv', 'client\_identifier');

*-- If no client\_identifier is available, we'll have to give it something, so grab*

*-- identifying information from the data dictionary.*

**IF** (gr\_client\_ctx.client\_id **IS** **NULL**) **THEN**

gr\_client\_ctx.client\_id := **SUBSTR**(get\_client\_host||':'||get\_client\_program||':'||get\_client\_os\_user,1,255);

**END** **IF**;

**END** **IF**;

**RETURN** gr\_client\_ctx.client\_id;

**END** get\_client\_id;

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

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

**BEGIN**

**IF** (gr\_client\_ctx.client\_ip **IS** **NULL**) **THEN**

gr\_client\_ctx.client\_ip := **SYS\_CONTEXT**('userenv', 'ip\_address');

**END** **IF**;

**RETURN** gr\_client\_ctx.client\_ip;

**END** get\_client\_ip;

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

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

**BEGIN**

**IF** (gr\_client\_ctx.client\_host **IS** **NULL**) **THEN**

gr\_client\_ctx.client\_host := **SYS\_CONTEXT**('userenv', 'host');

**END** **IF**;

**RETURN** gr\_client\_ctx.client\_host;

**END** get\_client\_host;

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

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

**BEGIN**

**IF** (gr\_client\_ctx.client\_os\_user **IS** **NULL**) **THEN**

gr\_client\_ctx.client\_os\_user := **SYS\_CONTEXT**('userenv', 'os\_user');

**END** **IF**;

**RETURN** gr\_client\_ctx.client\_os\_user;

**END** get\_client\_os\_user;

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

**FUNCTION** get\_client\_program **RETURN** **VARCHAR2** **IS**

**BEGIN**

**IF** (gr\_client\_ctx.client\_program **IS** **NULL**) **THEN**

**SELECT** **program**

**INTO** gr\_client\_ctx.client\_program

**FROM** v$session

**WHERE** **sid** = get\_sid;

**END** **IF**;

**RETURN** gr\_client\_ctx.client\_program;

**END** get\_client\_program;

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

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

**BEGIN**

**IF** (gr\_client\_ctx.client\_module **IS** **NULL**) **THEN**

**IF** (get\_db\_version < 10) **THEN**

**SELECT** **MODULE**

**INTO** gr\_client\_ctx.client\_module

**FROM** v$session

**WHERE** **sid** = get\_sid;

**ELSE**

gr\_client\_ctx.client\_module := **SYS\_CONTEXT**('userenv', 'module');

**END** **IF**;

**END** **IF**;

**RETURN** gr\_client\_ctx.client\_module;

**END** get\_client\_module;

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

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

**BEGIN**

**IF** (gr\_client\_ctx.client\_action **IS** **NULL**) **THEN**

**IF** (get\_db\_version < 10) **THEN**

**SELECT** action

**INTO** gr\_client\_ctx.client\_action

**FROM** v$session

**WHERE** **sid** = get\_sid;

**ELSE**

gr\_client\_ctx.client\_action := **SYS\_CONTEXT**('userenv', 'action');

**END** **IF**;

**END** **IF**;

**RETURN** gr\_client\_ctx.client\_action;

**END** get\_client\_action;

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

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

**BEGIN**

**IF** (gr\_client\_ctx.session\_user **IS** **NULL**) **THEN**

gr\_client\_ctx.session\_user := **SYS\_CONTEXT**('userenv', 'session\_user');

**END** **IF**;

**RETURN** gr\_client\_ctx.session\_user;

**END** get\_session\_user;

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

**FUNCTION** get\_current\_schema **RETURN** **VARCHAR2** **IS**

**BEGIN**

**IF** (gr\_client\_ctx.current\_schema **IS** **NULL**) **THEN**

*-- The current\_schema can be statically set by calling set\_current\_schema. If*

*-- that hasn't been called, then the current\_schema will be obtained from the*

*-- data dictionary for the current execution context, and placed in the local*

*-- memory structure and application context.*

set\_current\_schema(**SYS\_CONTEXT**('userenv', 'current\_schema'));

**END** **IF**;

**RETURN** gr\_client\_ctx.current\_schema;

**END** get\_current\_schema;

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

**FUNCTION** get\_db\_version **RETURN** **NUMBER** **IS**

*-- l\_version VARCHAR2(30);*

**BEGIN**

*/\**

*NOTE: The entire body of this routine could be replaced by proper reference*

*to dbms\_db\_version.version if your shop has only 10g or higher Oracle. 9i*

*is also possible if dbms\_db\_version was installed properly.*

*Could also use dbms\_utility.db\_version which returns two parameters, the*

*version string and compatiblity version.*

*\*/*

*-- IF (gr\_server\_ctx.db\_version IS NULL) THEN*

*--*

*-- SELECT SUBSTR(banner,*

*-- instr(banner, chr(9), 1, 1) + 1,*

*-- instr(banner, chr(9), 1, 2) - instr(banner, chr(9), 1, 1) - 1) version*

*-- INTO l\_version*

*-- FROM v$version*

*-- WHERE upper(banner) LIKE 'CORE%';*

*--*

*-- IF (l\_version LIKE '9%') THEN*

*-- gr\_server\_ctx.db\_version := 9;*

*-- ELSIF (l\_version LIKE '10%') THEN*

*-- gr\_server\_ctx.db\_version := 10;*

*-- ELSIF (l\_version LIKE '11%') THEN*

*-- gr\_server\_ctx.db\_version := 11;*

*-- ELSIF (l\_version LIKE '8%') THEN*

*-- gr\_server\_ctx.db\_version := 8;*

*-- ELSIF (l\_version LIKE '7%') THEN*

*-- gr\_server\_ctx.db\_version := 7;*

*-- END IF;*

*--*

*-- END IF;*

*--*

**IF** (gr\_server\_ctx.db\_version **IS** **NULL**) **THEN**

gr\_server\_ctx.db\_version := dbms\_db\_version.version;

**END** **IF**;

**RETURN** gr\_server\_ctx.db\_version;

**END** get\_db\_version;

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

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

**BEGIN**

**IF** (gr\_server\_ctx.db\_name **IS** **NULL**) **THEN**

gr\_server\_ctx.db\_name := **SYS\_CONTEXT**('userenv', 'db\_name');

**END** **IF**;

**RETURN** gr\_server\_ctx.db\_name;

**END** get\_db\_name;

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

**FUNCTION** get\_db\_instance\_name **RETURN** **VARCHAR2** **IS**

l\_instance\_name v$instance.instance\_name%**TYPE**;

lx\_catalog\_invisible **EXCEPTION**;

**PRAGMA** **EXCEPTION\_INIT**(lx\_catalog\_invisible,-942);

**BEGIN**

**IF** (get\_db\_version < 10) **THEN**

**SELECT** instance\_name

**INTO** l\_instance\_name

**FROM** v$instance

**WHERE** instance\_number = **SYS\_CONTEXT**('userenv', 'instance');

**ELSE**

l\_instance\_name := **SYS\_CONTEXT**('userenv', 'instance\_name');

**END** **IF**;

**RETURN** l\_instance\_name;

**EXCEPTION**

**WHEN** lx\_catalog\_invisible **OR** **NO\_DATA\_FOUND** **THEN**

**RETURN** cnst.UNKNOWN\_STR;

**END** get\_db\_instance\_name;

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

**FUNCTION** get\_db\_instance\_id **RETURN** **NUMBER** **IS**

l\_instance\_id **NUMBER**;

**BEGIN**

*-- v$instance.inst\_id and instance\_number look exactly the same. I'm unsure*

*-- if and when they would ever differ within a given RAC cluster.*

*-- dbms\_utility.current\_instance seems to provide similar functionality*

l\_instance\_id := **SYS\_CONTEXT**('userenv', 'instance');

**RETURN** l\_instance\_id;

**END** get\_db\_instance\_id;

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

**FUNCTION** get\_server\_host **RETURN** **VARCHAR2** **IS**

l\_host\_name v$instance.host\_name%**TYPE**;

lx\_catalog\_invisible **EXCEPTION**;

**PRAGMA** **EXCEPTION\_INIT**(lx\_catalog\_invisible,-942);

**BEGIN**

**IF** (get\_db\_version < 10) **THEN**

**SELECT** host\_name

**INTO** l\_host\_name

**FROM** v$instance

**WHERE** instance\_number = **SYS\_CONTEXT**('userenv', 'instance');

**ELSE**

l\_host\_name := **SYS\_CONTEXT**('userenv', 'server\_host');

**END** **IF**;

**RETURN** l\_host\_name;

**EXCEPTION**

**WHEN** lx\_catalog\_invisible **OR** **NO\_DATA\_FOUND** **THEN**

**RETURN** cnst.unknown\_str;

**END** get\_server\_host;

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

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

*--BEGIN*

*-- RETURN SYS\_CONTEXT('userenv', 'global\_context\_memory');*

*--END get\_global\_context\_memory;*

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

**FUNCTION** get\_sid **RETURN** **INTEGER** **IS**

**BEGIN**

**RETURN** get\_session\_id;

**END** get\_sid;

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

**FUNCTION** get\_session\_id **RETURN** **INTEGER** **IS**

l\_sid **INTEGER** := 0;

**BEGIN**

**IF** (get\_db\_version < 10) **THEN**

**SELECT** **DISTINCT** **sid**

**INTO** l\_sid

**FROM** v$mystat;

**ELSE**

l\_sid := **SYS\_CONTEXT**('userenv', 'sid');

**END** **IF**;

**RETURN** l\_sid;

**END** get\_session\_id;

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

**FUNCTION** get\_os\_pid **RETURN** **INTEGER**

**IS**

l\_pid **NUMBER** := 0;

lx\_catalog\_invisible **EXCEPTION**;

**PRAGMA** **EXCEPTION\_INIT**(lx\_catalog\_invisible,-942);

**BEGIN**

**EXECUTE** **IMMEDIATE** '

SELECT p.spid

FROM v$session s,

v$process p

WHERE s.audsid = SYS\_CONTEXT(''userenv'', ''sessionid'')

AND s.paddr = p.addr '

**INTO** l\_pid;

**RETURN** l\_pid;

**EXCEPTION**

**WHEN** lx\_catalog\_invisible **OR** **NO\_DATA\_FOUND** **THEN**

**RETURN** l\_pid; *-- just return zero*

*-- WHEN TOO\_MANY\_ROWS THEN*

*-- ins('env.get\_os\_pid found more than one row.');*

*-- RETURN l\_pid;*

**END** get\_os\_pid;

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

**FUNCTION** get\_schema\_email\_address **RETURN** **VARCHAR2**

**IS**

**BEGIN**

**RETURN** get\_current\_schema||'\_'||get\_db\_name||'@'||get\_server\_host||'.'||DOMAIN;

**END** get\_schema\_email\_address;

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

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

**IS**

l\_db\_id app\_env.db\_id%**TYPE**;

**BEGIN**

**SELECT** db\_id

**INTO** l\_db\_id

**FROM** app\_db

**WHERE** **LOWER**(db\_nm) = **LOWER**(get\_db\_name);

**RETURN** l\_db\_id;

**END** get\_db\_id;

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

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

l\_db\_alias app\_db.db\_alias%**TYPE**;

**BEGIN**

**IF** (**UPPER**(i\_db) **LIKE** 'MSSW%') **THEN**

l\_db\_alias := **LOWER**(i\_db);

**ELSE**

**SELECT** **LOWER**(db\_alias)

**INTO** l\_db\_alias

**FROM** app\_db

**WHERE** db\_nm = **NVL**(**UPPER**(i\_db),**UPPER**(env.get\_db\_name))

**OR** db\_alias = **UPPER**(i\_db);

**END** **IF**;

**RETURN** l\_db\_alias;

**EXCEPTION**

**WHEN** **NO\_DATA\_FOUND** **THEN**

RAISE\_APPLICATION\_ERROR(-20000, i\_db || ' is not a valid database SID, service name or alias.');

**END** get\_db\_alias;

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

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

l\_app\_id app.app\_id%**TYPE**;

l\_app\_cd app.app\_cd%**TYPE**;

**BEGIN**

**IF** (i\_app\_cd **IS** **NULL**) **THEN**

*-- Here the app\_cd is missing. So we are assuming the caller wants us to*

*-- determine the application dynamically using the application context or*

*-- data dictionary (data as set up in APP, APP\_DB, and APP\_ENV).*

**IF** (gr\_client\_ctx.app\_id **IS** **NULL**) **THEN**

*--ins('get\_app\_id: gr\_client\_ctx.app\_id is NULL');*

*-- Attempt to get app code from the default context. Could be set by*

*-- an after logon trigger.*

l\_app\_cd := **SYS\_CONTEXT**(app\_core\_ctx, 'app\_cd');

*--ins('get\_app\_id: app\_cd in app\_core\_ctx is ['||l\_app\_cd||']');*

**IF** (l\_app\_cd **IS** **NOT** **NULL**) **THEN**

**SELECT** app\_id

**INTO** l\_app\_id

**FROM** app

**WHERE** **LOWER**(app\_cd) = **LOWER**(l\_app\_cd);

**ELSE**

**BEGIN**

**SELECT** app\_id

**INTO** l\_app\_id

**FROM** app\_env aev,

app\_db adb

**WHERE** adb.db\_nm = **UPPER**(**SYS\_CONTEXT**('userenv', 'db\_name'))

**AND** aev.db\_id = adb.db\_id

**AND** (

aev.access\_account = **SYS\_CONTEXT**('userenv', 'session\_user')

**OR**

aev.owner\_account = **SYS\_CONTEXT**('userenv', 'current\_schema')

);

**EXCEPTION**

**WHEN** **TOO\_MANY\_ROWS** **THEN**

*-- Multiple matches will happen in access and object-owning*

*-- accounts when multiple applications run out of the same schema.*

*-- We will try to get the app\_id one more time from APP\_DB and*

*-- APP\_ENV using just the access\_account. After that, if we still have*

*-- too many rows, then the caller is the owning schema that has*

*-- the multiple apps, which will require the app\_cd set in context*

*-- in order to resolve. If the app\_cd has not been set in context,*

*-- we'll raise an error.*

**BEGIN**

**SELECT** app\_id

**INTO** l\_app\_id

**FROM** app\_env aev,

app\_db adb

**WHERE** adb.db\_nm = **UPPER**(**SYS\_CONTEXT**('userenv', 'db\_name'))

**AND** aev.db\_id = adb.db\_id

**AND** aev.access\_account = **SYS\_CONTEXT**('userenv', 'session\_user');

**EXCEPTION**

**WHEN** **NO\_DATA\_FOUND** **OR** **TOO\_MANY\_ROWS** **THEN**

RAISE\_APPLICATION\_ERROR(-20000, 'Unable to determine '||

'application from environment, Core tables or '||

env.app\_core\_ctx||'.app\_cd ['||

**NVL**(**SYS\_CONTEXT**(app\_core\_ctx, 'app\_cd'),'Not Set')||'].');

**END**; *-- second attempt using just access account*

**END**; *-- first attempt using owner or access account*

**END** **IF**;

*-- One of the three SELECT statements above should have found it by now.*

*-- Store in package structure that is kept in memory. Future calls to*

*-- env.get\_app\_id will be very quick as only this struct will be read.*

gr\_client\_ctx.app\_id := l\_app\_id;

**ELSE**

l\_app\_id := gr\_client\_ctx.app\_id;

**END** **IF**;

**ELSE**

*-- Here we are given the app\_cd, so we'll just do a simple lookup and not*

*-- assume the caller wants it stored in the session's UGA.*

**SELECT** app\_id

**INTO** l\_app\_id

**FROM** app

**WHERE** **LOWER**(app\_cd) = **LOWER**(i\_app\_cd);

**END** **IF**;

**RETURN** l\_app\_id;

*-- We do not handle NO\_DATA\_FOUND on purpose so that the error bubbles up,*

*-- clearly indicating that data and/or schemas aren't set up properly.*

**END** get\_app\_id;

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

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

l\_app\_cd app.app\_cd%**TYPE**;

**BEGIN**

**IF** (i\_app\_id **IS** **NOT** **NULL**) **THEN**

**SELECT** app\_cd

**INTO** l\_app\_cd

**FROM** app

**WHERE** app\_id = i\_app\_id;

**ELSE**

**SELECT** app\_cd

**INTO** l\_app\_cd

**FROM** app

**WHERE** app\_id = get\_app\_id;

**END** **IF**;

**RETURN** l\_app\_cd;

*-- We do not handle NO\_DATA\_FOUND on purpose so that the error bubbles up,*

*-- clearly indicating that data and/or schemas aren't set up properly.*

**END** get\_app\_cd;

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

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

l\_env\_nm app\_env.env\_nm%**TYPE**;

l\_app\_id app\_env.app\_id%**TYPE**;

**BEGIN**

*-- i\_app\_cd may be empty, which will cause get\_app\_id to get app\_cd dynamically*

l\_app\_id := get\_app\_id(i\_app\_cd);

*-- Get environment name by app\_id*

**BEGIN**

**SELECT** env\_nm

**INTO** l\_env\_nm

**FROM** app\_env

**WHERE** app\_id = l\_app\_id

**AND** db\_id = get\_db\_id

**AND** owner\_account = get\_current\_schema;

**EXCEPTION**

**WHEN** **NO\_DATA\_FOUND** **THEN**

l\_env\_nm := cnst.UNKNOWN\_STR;

**END**;

**RETURN** l\_env\_nm;

**END** get\_env\_nm;

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

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

**IS**

l\_dir\_path all\_directories.directory\_path%**TYPE**;

**BEGIN**

**SELECT** directory\_path

**INTO** l\_dir\_path

**FROM** all\_directories

**WHERE** directory\_name = **UPPER**(**TRIM**(i\_dir\_nm));

**RETURN** l\_dir\_path;

**EXCEPTION**

**WHEN** **NO\_DATA\_FOUND** **THEN**

**RETURN** 'Directory '||i\_dir\_nm||' Not Found';

**END** get\_dir\_path;

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

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

**IS**

**BEGIN**

**IF** (**SUBSTR**(i\_path,**LENGTH**(i\_path),1) **IN** ('/','\')) **THEN**

**RETURN** i\_path;

**ELSE**

**RETURN** i\_path||cnst.DIR\_SEPCHAR;

**END** **IF**;

**END** vld\_path\_format;

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

**FUNCTION** who\_called\_me (

i\_stack\_level **IN** **PLS\_INTEGER** **DEFAULT** 2

) **RETURN** **VARCHAR2**

**IS**

lar\_stack\_data tar\_stack\_data;

l\_my\_name app\_log.routine\_nm%**TYPE**;

**BEGIN**

bundle\_stack\_lines(lar\_stack\_data);

**IF** (i\_stack\_level <= lar\_stack\_data.COUNT) **THEN**

**IF** (**UPPER**(lar\_stack\_data(i\_stack\_level).obj\_type) = 'PACKAGE BODY') **THEN**

l\_my\_name := get\_routine\_nm(lar\_stack\_data(i\_stack\_level).obj\_nm,

lar\_stack\_data(i\_stack\_level).line\_num);

**ELSE**

l\_my\_name := lar\_stack\_data(i\_stack\_level).obj\_nm;

**END** **IF**;

**ELSE**

l\_my\_name := cnst.UNKNOWN\_STR;

**END** **IF**;

**RETURN** l\_my\_name;

**END** who\_called\_me;

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

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

lar\_stack\_data tar\_stack\_data;

l\_my\_name app\_log.routine\_nm%**TYPE**;

**BEGIN**

bundle\_stack\_lines(lar\_stack\_data);

**IF** (i\_stack\_level <= lar\_stack\_data.COUNT) **THEN**

**IF** (**UPPER**(lar\_stack\_data(i\_stack\_level).obj\_type) = 'PACKAGE BODY') **THEN**

l\_my\_name := get\_routine\_nm(lar\_stack\_data(i\_stack\_level).obj\_nm,

lar\_stack\_data(i\_stack\_level).line\_num);

**ELSE**

l\_my\_name := lar\_stack\_data(i\_stack\_level).obj\_nm;

**END** **IF**;

**ELSE**

l\_my\_name := cnst.UNKNOWN\_STR;

**END** **IF**;

**RETURN** l\_my\_name;

**END** who\_am\_i;

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

**FUNCTION** get\_routine\_nm

(

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

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

) **RETURN** **VARCHAR2**

**IS**

l\_routine\_nm app\_log.routine\_nm%**TYPE**;

**BEGIN**

**SELECT** package\_name || '.' || routine\_name

**INTO** l\_routine\_nm

**FROM** *-- Use RANK to get the containing routine, the one closest to the given line number*

(**SELECT** t.package\_name,

t.routine\_name,

t.start\_line,

**RANK**() **OVER**(**ORDER** **BY** t.start\_line **DESC**) rnk

**FROM** *-- Use TRANSLATE to parse the declarations into just the routine names, so they can be joined*

*-- to USER\_ARGUMENTS to eliminate inner (nested) routines.*

(

**SELECT** **name** **AS** package\_name,

line **AS** start\_line,

**RTRIM**(**SUBSTR**(after\_token,1,**INSTR**(**TRANSLATE**(after\_token,**CHR**(13)||**CHR**(10)||**CHR**(32)||'(/-','++++++'),'+')-1)) routine\_name,

text **AS** orig\_src\_line

**FROM** *-- Get all the routine declarations that came before the caller's line. One of them*

*-- must be the routine that contains the line.*

(**SELECT** **name**,

line,

text,

**DECODE**(**SIGN**(**INSTR**(**UPPER**(text), 'PROCEDURE')),

1,

**TRIM**(**SUBSTR**(**UPPER**(text), **INSTR**(**UPPER**(text), 'PROCEDURE') + 10)),

**TRIM**(**SUBSTR**(**UPPER**(text), **INSTR**(**UPPER**(text), 'FUNCTION') + 9))

) after\_token

**FROM** user\_source

**WHERE** **type** = 'PACKAGE BODY'

**AND** **name** = **UPPER**(i\_pkg\_nm)

**AND** line <= i\_line\_num *-- only look at code before the caller's line number*

**AND** (**UPPER**(**TRIM**(text)) **LIKE** 'PROCEDURE%' **OR** **UPPER**(**TRIM**(text)) **LIKE** 'FUNCTION%'))

) t

*-- DISTINCT and analytic ROW\_NUMBER yielded same response time, so went with more readable SQL*

*-- 2010Mar30: Unfortunately, this hides private routines as well, so going back to behavior*

*-- where inner routines mess it up.*

*-- ,(SELECT DISTINCT package\_name, object\_name FROM user\_arguments) ua*

*-- WHERE ua.package\_name = t.package\_name*

*-- AND t.routine\_name = ua.object\_name -- eliminates inner routines*

)

**WHERE** rnk = 1;

**RETURN** l\_routine\_nm;

**EXCEPTION**

**WHEN** **NO\_DATA\_FOUND** **THEN**

**RETURN** **NULL**;

**END** get\_routine\_nm;

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

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

**IS**

lar\_stack\_data tar\_stack\_data;

**BEGIN**

bundle\_stack\_lines(lar\_stack\_data);

**IF** (i\_stack\_level <= lar\_stack\_data.COUNT) **THEN**

**RETURN** lar\_stack\_data(i\_stack\_level).line\_num;

**ELSE**

**RETURN** **NULL**;

**END** **IF**;

**END** line\_num\_here;

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

**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

) **IS**

lar\_stack\_data tar\_stack\_data;

**BEGIN**

bundle\_stack\_lines(lar\_stack\_data);

*--ins('caller\_meta: lar\_stack\_data.COUNT = '||lar\_stack\_data.COUNT);*

**IF** (i\_stack\_level <= lar\_stack\_data.COUNT) **THEN**

o\_owner := lar\_stack\_data(i\_stack\_level).owner;

o\_caller\_type := lar\_stack\_data(i\_stack\_level).obj\_type;

o\_unit\_nm := lar\_stack\_data(i\_stack\_level).obj\_nm;

*-- get underlying procedure/function name if called from package*

**IF** (**UPPER**(lar\_stack\_data(i\_stack\_level).obj\_type) = 'PACKAGE BODY') **THEN**

*--ins('caller\_meta: calling get\_routine\_nm('''||lar\_stack\_data(i\_stack\_level).obj\_nm||*

*-- ''','||lar\_stack\_data(i\_stack\_level).line\_num||')');*

o\_routine\_nm := get\_routine\_nm(lar\_stack\_data(i\_stack\_level).obj\_nm,

lar\_stack\_data(i\_stack\_level).line\_num);

**ELSE**

o\_routine\_nm := lar\_stack\_data(i\_stack\_level).obj\_nm;

**END** **IF**;

o\_line\_num := lar\_stack\_data(i\_stack\_level).line\_num;

**ELSE**

o\_owner := cnst.unknown\_user;

o\_caller\_type := **NULL**;

o\_unit\_nm := cnst.unknown\_str;

o\_routine\_nm := cnst.unknown\_str;

o\_line\_num := **NULL**;

**END** **IF**;

**END** caller\_meta;

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

**PROCEDURE** tag\_session

(

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

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

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

)

**IS**

**BEGIN**

*-- We do not use excp.throw here so ENV can maintain low-level independence*

**IF** (**LENGTH**(i\_module) > 48) **THEN**

RAISE\_APPLICATION\_ERROR(-20000, 'ERROR: (Assertion Failure) [env.'||who\_am\_i||']'||

' i\_module must be 48 characters or less');

**ELSIF** (**LENGTH**(i\_action) > 32) **THEN**

RAISE\_APPLICATION\_ERROR(-20000, 'ERROR: (Assertion Failure) [env.'||who\_am\_i||']'||

': i\_action must be 32 characters or less');

**ELSIF** (**LENGTH**(i\_info) > 64) **THEN**

RAISE\_APPLICATION\_ERROR(-20000, 'ERROR: (Assertion Failure) [env.'||who\_am\_i||']'||

': i\_info must be 64 characters or less');

**END** **IF**;

push\_trace\_stack(i\_module, i\_action, i\_info);

dbms\_application\_info.set\_module(i\_module, i\_action);

dbms\_application\_info.set\_client\_info(i\_info);

**END** tag\_session;

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

**PROCEDURE** tag

(

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

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

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

)

**IS**

l\_module v$session.module%**TYPE** := i\_module;

l\_action v$session.action%**TYPE** := i\_action;

l\_info v$session.client\_info%**TYPE** := i\_info;

**BEGIN**

**IF** (l\_module **IS** **NULL** **OR** l\_action **is** **NULL** **OR** l\_info **IS** **NULL**) **THEN**

**DECLARE**

l\_unit\_nm typ.t\_maxobjnm;

l\_owner typ.t\_maxobjnm;

l\_caller\_type user\_objects.object\_type%**TYPE**;

l\_routine\_nm app\_log.routine\_nm%**TYPE**;

l\_line\_num app\_log.line\_num%**TYPE**;

**BEGIN**

env.caller\_meta(l\_owner, l\_caller\_type, l\_unit\_nm, l\_routine\_nm, l\_line\_num, 1);

**IF** (i\_module **IS** **NULL**) **THEN**

l\_module := l\_unit\_nm;

**END** **IF**;

**IF** (i\_action **IS** **NULL**) **THEN**

*-- substr to eliminate package name from routine name, dot-notation found*

l\_action := **SUBSTR**(l\_routine\_nm,**INSTR**(l\_routine\_nm,'.')+1);

**END** **IF**;

**IF** (i\_info **IS** **NULL**) **THEN**

l\_info := l\_line\_num;

**END** **IF**;

**END**;

**END** **IF**;

tag\_session(l\_module, l\_action, l\_info);

**END** tag;

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

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

l\_module v$session.module%**TYPE** := **NULL**;

l\_action v$session.action%**TYPE** := **NULL**;

l\_info v$session.client\_info%**TYPE** := **NULL**;

**BEGIN**

pop\_trace\_stack;

**IF** (i\_restore\_prior\_tag) **THEN**

**IF** (g\_trace\_stack.COUNT > 0) **THEN**

l\_module := g\_trace\_stack(g\_trace\_stack.COUNT).module;

l\_action := g\_trace\_stack(g\_trace\_stack.COUNT).action;

l\_info := g\_trace\_stack(g\_trace\_stack.COUNT).client\_info;

**END** **IF**;

**END** **IF**;

dbms\_application\_info.set\_client\_info(l\_info);

dbms\_application\_info.set\_module(l\_module, l\_action);

**END** untag\_session;

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

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

**BEGIN**

untag\_session(i\_restore\_prior\_tag);

**END** untag;

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

**PROCEDURE** set\_ctx\_val

(

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

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

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

)

**IS**

**BEGIN**

dbms\_session.set\_context(i\_ctx\_nm, i\_attr\_nm, i\_attr\_val);

**END** set\_ctx\_val;

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

**PROCEDURE** set\_app\_cd

(

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

)

**IS**

**BEGIN**

**IF** (i\_app\_cd **IS** **NOT** **NULL**) **THEN**

*--ins('set\_app\_cd: setting app\_id for app\_cd '||i\_app\_cd);*

gr\_client\_ctx.app\_id := get\_app\_id(i\_app\_cd);

set\_ctx\_val('app\_cd',i\_app\_cd);

**ELSE**

RAISE\_APPLICATION\_ERROR(-20000,'ERROR: (Assertion Failure) [env.set\_app\_cd]'||

' Application Code cannot be empty.');

**END** **IF**;

**END** set\_app\_cd;

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

**PROCEDURE** set\_current\_schema

(

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

)

**IS**

**BEGIN**

**IF** (i\_schema\_nm **IS** **NOT** **NULL**) **THEN**

*--ins('set\_current\_schema: setting current\_schema to '||i\_schema\_nm);*

gr\_client\_ctx.current\_schema := **UPPER**(i\_schema\_nm);

set\_ctx\_val('current\_schema',**UPPER**(i\_schema\_nm));

**ELSE**

RAISE\_APPLICATION\_ERROR(-20000,'ERROR: (Assertion Failure) [env.set\_current\_schema]'||

' Schema Name cannot be empty.');

**END** **IF**;

**END** set\_current\_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**

) **IS**

**BEGIN**

*-- Populate in-memory structure with new identifier. If not passed in*

gr\_client\_ctx.client\_id := i\_client\_id;

**IF** (i\_client\_ip **IS** **NOT** **NULL**) **THEN**

gr\_client\_ctx.client\_ip := i\_client\_ip;

**END** **IF**;

**IF** (i\_client\_host **IS** **NOT** **NULL**) **THEN**

gr\_client\_ctx.client\_host := i\_client\_host;

**END** **IF**;

**IF** (i\_client\_os\_user **IS** **NOT** **NULL**) **THEN**

gr\_client\_ctx.client\_os\_user := i\_client\_os\_user;

**END** **IF**;

**IF** (i\_app\_cd **IS** **NOT** **NULL**) **THEN**

set\_app\_cd(i\_app\_cd);

**END** **IF**;

**IF** (gr\_client\_ctx.current\_schema **IS** **NULL**) **THEN**

set\_current\_schema(**SYS\_CONTEXT**('userenv','current\_schema'));

**END** **IF**;

*-- Populate USERENV with the given client identifier*

dbms\_session.set\_identifier(i\_client\_id);

**END** init\_client\_ctx;

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

**PROCEDURE** reset\_client\_ctx **IS**

**BEGIN**

dbms\_session.clear\_identifier;

dbms\_session.modify\_package\_state(dbms\_session.reinitialize);

clear\_ctx; *-- clears Core framework context namespace*

gr\_client\_ctx := gr\_empty\_client\_ctx; *-- clears global variables for this package*

**END** reset\_client\_ctx;

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

**PROCEDURE** clear\_ctx\_val

(

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

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

)

**IS**

**BEGIN**

dbms\_session.clear\_context(i\_ctx\_nm, **NULL**, i\_attr\_nm);

**END** clear\_ctx\_val;

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

**PROCEDURE** clear\_ctx

(

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

)

**IS**

**BEGIN**

*-- 10g only*

*--dbms\_session.clear\_all\_context(i\_ctx\_nm);*

*-- clear\_all\_context doesn't exist in 9i*

dbms\_session.clear\_context(i\_ctx\_nm, **NULL**, **NULL**);

**END** clear\_ctx;

**END** env;