Well, as a very crude test you can set your authentication scheme to be Application Express Authenticationand define some users in Apex. Then create a before header login process on the login page (101) to automatically logon containing:

apex\_authentication.login(

    p\_username => :P101\_USERNAME,

    p\_password => :P101\_PASSWORD );

:P101\_PASSWORD := NULL;

conditional on P101\_PASSWORD is not null.  
  
Then you can login by specifying your username and password in the URL e.g.  
  
f?p=YOURAPP:LOGIN:::::P101\_USERNAME,P101\_PASSWORD:username,password

https://www.wegobeyond.co.uk/blog/extending-e-business-suite-with-oracle-application-express-apex

**Extending E-Business Suite with Oracle Application Express (APEX)**

[E-Business Suite](https://www.wegobeyond.co.uk/blog/categories/e_business_suite) [APEX](https://www.wegobeyond.co.uk/blog/categories/apex)

[John Keymer](https://www.wegobeyond.co.uk/blog/blogger/johnkeymer)

 Thursday, 12 March 2015

 27309 Hits

[2 Comments](https://www.wegobeyond.co.uk/blog/extending-e-business-suite-with-oracle-application-express-apex#comments)

Traditionally E-Business Suite projects have consisted of a number of technologies - Oracle Forms, OAF, Reports, XMLP, SQL\*PLUS and so on. Each of these has its own pro's and con's however most are reasonably time-tested and well integrated into the E-Business Suite technology stack.

Over recent years Oracle have put a lot of time and effort into the development of [Oracle Application Express (APEX)](http://www.oracle.com/technetwork/developer-tools/apex/overview/index.html#_blank) - Formerly HTMLDB - a rapid application development tool for developing web-based applications on the Oracle database. APEX has truly come on leaps and bounds over recent years, which may surprise some people as it is FREE. Yes, **FREE**. Ok, that may not be *strictly* true; you still need an Oracle Database with which APEX now comes shipped, and if that is for commercial purposes then you need to licence that database. However you pay no additional licencing fees to use APEX within your existing infrastructure. If you want to just give it a try then you can get yourself a [free Oracle-hosted "workspace"](https://apex.oracle.com/i/#_blank), however this should be used strictly for development and proof of concept only as there is no guarantee of service availability etc. You can also download Oracle XE for free and upgrade to the latest APEX version. I have a small Slackware Linux VirtualBox machine with XE installed and the latest APEX release which I use as a sandbox for proof of concepts etc.  
For these reasons combined with the overheads of existing technologies and the very shallow learning curve for basic APEX functionality, more and more customers are looking at how they might utilize APEX for their custom E-Business Suite extensions. Whilst it is possible to simply install APEX and query/update tables (via the supported methods of course!) on the E-Business Suite database within a very quick space of time, what customers are generally looking for is **seamless integration**. That is, their APEX application is fully integrated within their E-Business Suite environment. Users should not even be aware that they are using a "different" technology. Furthermore, the advantages of this are of course:

* No need to manage users separately - users authenticate using their E-Business Suite credentials.
* No separate login pages or links - users log in using the E-Business Suite login page.
* No need for separate security models - application pages and functionality can be controlled using standard E-Business Suite functionality such as responsibilities, security profiles etc.
* Utilization of E-Business Suite context - profile values, fnd\_global constants etc.

There are three basic steps to integrating your APEX installation with your E-Business environment

1. [Technical Installation](http://docs.oracle.com/cd/E37097_01/install.42/e35123/toc.htm#_blank)
2. Authentication/Seamless Integration
3. Best Practices

I am not going to cover the installation - it is well documented on the link above (ensure you consult the appropriate documentation for the version you are installing). There are several possible options available depending on your infrastructure and E-Business Suite version (Embedded PLSQL Gateway (EPG), Glassfish server etc), however your DBA should be able to advise. Ultimately the EPG is the "quickest and easiest" however has some caveats that should be understood. More information is available from MOS and [this excellent diagram](http://www.oracle.com/technetwork/developer-tools/apex/apex-arch-086399.html#_blank).   
Authentication and integration shall be the subject of this blog post and I will be posting another blog in the near future which covers best practices for development. A lot of the existing how-to's on integration out there at the moment are very light on technical content so hopefully this will help bridge that gap.

For now, it shall be assumed you have an APEX installation with a workspace configured that has access to (either via the parsing schema or grants) to the appropriate E-Business Objects you wish to use. I will cover the benefits of parsing as apps or otherwise in my next blog post - assume APPS for now for simplicity.

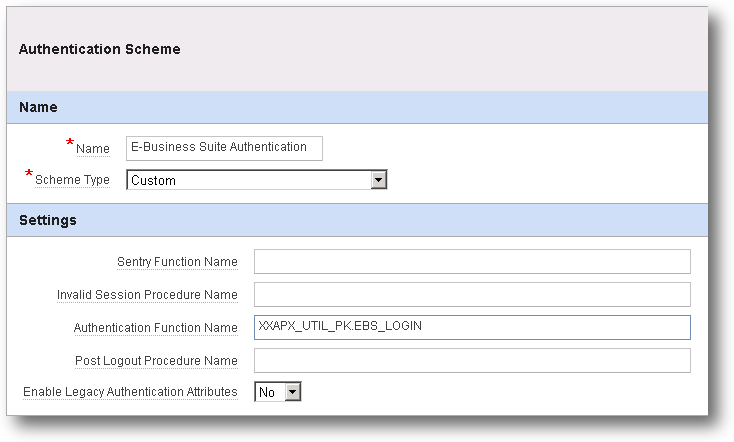
**Note - Parsing as APPS is not officially supported by Oracle**. The [whitepaper](http://www.oracle.com/technetwork/developer-tools/apex/learnmore/apex-ebs-extension-white-paper-345780.pdf#_blank) published by Oracle recommends that a minimal privilege schema be created and all objects granted to that. I have my own views on the justification for this which I have expressed to the author, however you should take your own decision on this. If you choose to parse as a custom schema the APEX side of things is exactly the same, however you will need to create grants to your custom schema and then synonyms as that user.

There are several possible ways of carrying out this integration, however this is the method I would recommend. It is best practice to hold as much code as possible on the database (i.e. in database packages) and for the purposes of this post I will assume that package is called **XXAPX\_UTIL\_PK***(see bottom of this post for sourcecode)*. Feel free to use your own naming conventions where you wish for both the package and any custom functions.

**This blog assumes you have a reasonable understanding on both E-Business Suite and APEX authentication/authorization.**  
If you need any further elaboration on the points then please ask in the comments.

So; you have created the basics of a new application and now want to integrate it.

1. Create a new Authentication Scheme called *E-Business Suite Authentication* with a scheme type of "Custom".
2. Set the authentication function to be XXAPX\_UTIL\_PK.EBS\_LOGIN
3. Switch your application to use this scheme



The login function basically just verifies there is a valid session. You can extend it to do more if you wish.

function ebs\_login(p\_username varchar2, p\_password varchar2) return boolean

is

l\_valid\_session boolean;

l\_session\_id icx\_sessions.session\_id%type;

cursor c\_icx is

select icxs.session\_id

from icx\_sessions icxs

where icxs.session\_id = icx\_sec.getsessioncookie

and icxs.last\_connect + numtodsinterval(icxs.time\_out,'MINUTE') > sysdate;

begin

open c\_icx;

fetch c\_icx into l\_session\_id;

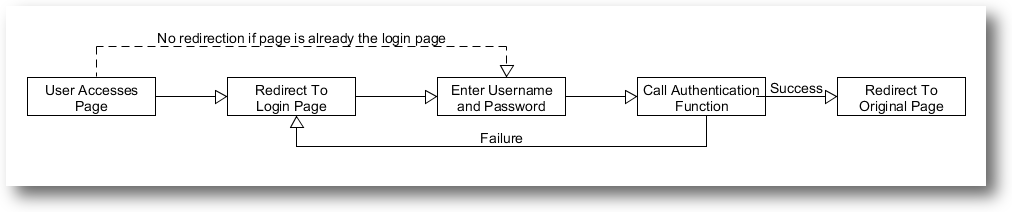
l\_valid\_session := c\_icx%found;

close c\_icx;

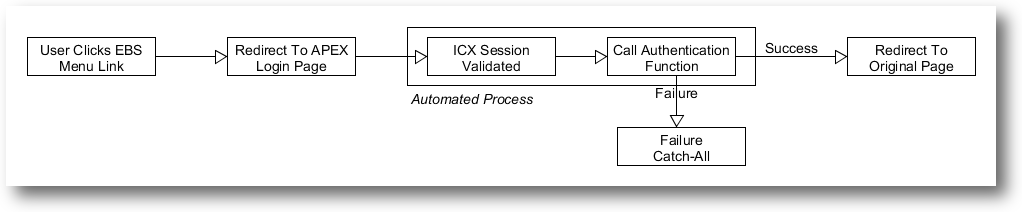
return l\_valid\_session;

end ebs\_login;

Now, the standard process is as follows:



In short, the user visits a page which requires authentication, the APEX engine redirects to the login page, the user enters their details, the authentication function is executed and the user is directed back to the page they wanted.  
However for E-Business Suite authentication we have already had the user enter a username and password - so we shouldn't be asking them to do it again. What we have is the following:



So a user visits a page as above and is redirected to the login page. However in this instance the login page automatically processes the login and finally redirects the user to where they wish to go. So how do we do that? Well first we create a new **Before Header** PLSQL process on the login page (traditionally page 101). As always, we hold as much code in the database as possible, so we will just make a call to a procedure in our package XXAPX\_UTIL\_PK. The key thing we do here is validate the user session first then call the custom authentication scheme we created above so APEX can set up the internal state.

At this point we need to consider one major feature of APEX which is that it is **stateless**. It does not retain the same database session throughout the lifecycle of application usage. So when you request a page a connection is made to the database at that point, the page is rendered and the session is disconnected. Subsequent pages similarly follow suit. They key thing here though is that the database **session id** changes between page requests. This can be seen by creating a simple report of

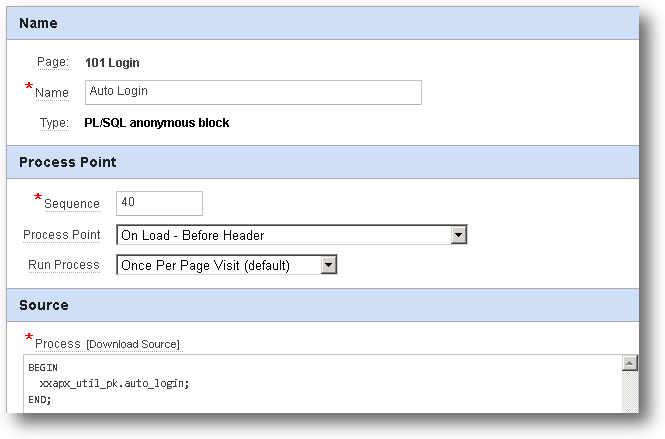
select userenv('sessionid') s from dual;

Then refresh the page a few times.  
The reason this is important is that **E-Business Context** is set up against a session. So if the session changes you lose your context! We therefore need to ensure that APEX can initiate the EBS context independently of its parent session. To do that we will create four application items and populate them at login with the following:

* User ID
* Responsibility ID
* Responsibility Application ID
* Security Group ID



So back onto our process, we create something along the lines of:



Now let's look at what processing we do in that procedure. We need to set up the E-Business Suite context which we can retrieve from the ICX Session cookie we saw earlier.

procedure auto\_login

is

l\_user\_id icx\_sessions.user\_id%type;

l\_resp\_id icx\_sessions.responsibility\_id%type;

l\_resp\_appl\_id icx\_sessions.responsibility\_application\_id%type;

l\_security\_group\_id icx\_sessions.security\_group\_id%type;

l\_user\_name fnd\_user.user\_name%type;

-- get the context information from the icx session cookie.

cursor c\_session

is

select s.user\_id,

s.responsibility\_id,

s.responsibility\_application\_id,

s.security\_group\_id,

u.user\_name

from icx\_sessions s, fnd\_user u

where u.user\_id = s.user\_id

and s.session\_id = icx\_sec.getsessioncookie

and s.last\_connect + numtodsinterval (s.time\_out, 'minute') > sysdate;

begin

open c\_session;

fetch c\_session

into l\_user\_id,

l\_resp\_id,

l\_resp\_appl\_id,

l\_security\_group\_id,

l\_user\_name;

close c\_session;

apex\_util.set\_session\_state ('GBL\_USER\_ID', l\_user\_id);

apex\_util.set\_session\_state ('GBL\_RESP\_ID', l\_resp\_id);

apex\_util.set\_session\_state ('GBL\_RESP\_APPL\_ID', l\_resp\_appl\_id);

apex\_util.set\_session\_state ('GBL\_SECURITY\_GROUP\_ID', l\_security\_group\_id);

set\_apps\_context; -- Explained below

-- call the standard apex login procedure to initialize a valid apex session.

wwv\_flow\_custom\_auth\_std.

login (p\_uname => l\_user\_name,

p\_password => '',

p\_session\_id => v ('app\_session'),

p\_flow\_page => v ('app\_alias') || ':1');

end auto\_login;

Naturally you will probably want to extend the above to take the next page as a parameter rather than being hard-coded to 1, possibly remove hard-coding of item names and so on... Anyway, in the above code you will see a procedure call to **set\_apps\_context**. This has been separated out intentionally because we use it elsewhere. This has the following content:

procedure set\_apps\_context

is

begin

fnd\_global.apps\_initialize (

user\_id => v('GBL\_USER\_ID'),

resp\_id => v('GBL\_RESP\_ID'),

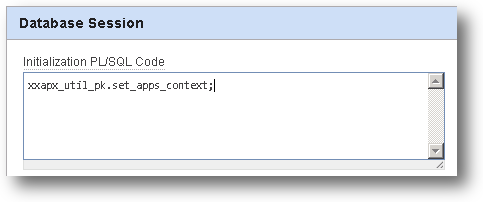
resp\_appl\_id => v('GBL\_RESP\_APPL\_ID'),

security\_group\_id => v('GBL\_SECURITY\_GROUP\_ID'));

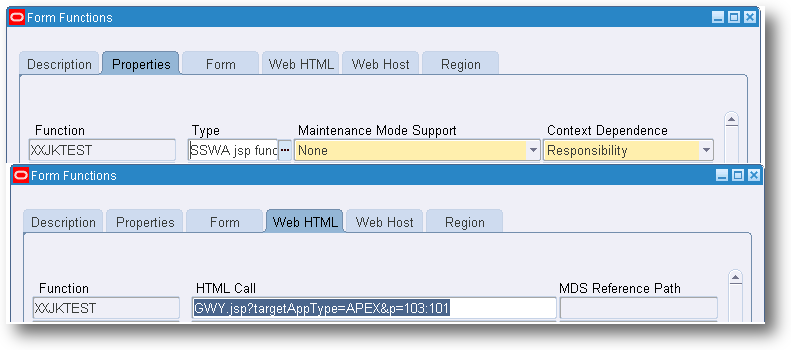
end set\_apps\_context;

Again, one would ideally remove the hard-coding from above and parameterize the item names. You may also want to extend the authentication function to prevent cookie tampering. Again, we're covering the bare bones here - that might be a topic of a future post!

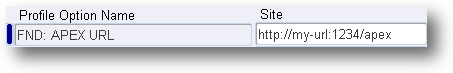
Finally we need to consider our earlier point on session state. Each time we do anything within our application we need to ensure the E-Business Suite context has been fully set up. To do this we utilize the APEX VPN functionality which allows a function or procedure to be specified that is executed before rendering.



And that's it from the APEX side. Now you just need to create the E-Business Suite function and add it to a menu. We utilize the existing Gateway (GWY.jsp) call (assuming EBS R12.x):



Finally ensure the **FND: APEX URL** profile value is set to the base URL of your APEX instance:



And that's everything! Now just log into E-Business Suite, select your responsibility and click the menu item. You should be launched directly into your APEX application, fully authenticated, and using E-Business Suite context.

There's a reasonable amount of information above, so here is a short checklist of key thigns you should have done.

1. Compiled your package code into the E-Business Suite instance as your **parsing schema**.
2. Created a new authentication scheme and assigned it to your application.
3. Created application items to hold the EBS context.
4. Added a new Before Header PLSQL process to the login page.
5. Added in context set code to the VPD section.

**You can download both the APEX application and database package I used from**[**here**](https://www.wegobeyond.co.uk/blog_resources/ebs2apex.zip)**, however anything you install into your system is done so entirely at your own risk.**

I will be writing a blog post in the future covering development topics such as:

* Best practices for coding
* Look and feel best practice
* Security considerations for integration
* Controlling access to different areas/functionality within the application
* Querying data from secured applications (such as Human Resources) - *although this might form a blog post in itself!*

So keep an eye out, and please feel free to comment or ask any questions.

# Integration Oracle Apex with E-Business Suite

Oracle Application Express (Oracle Apex) is a rapid web application development tool for the Oracle database. Using only a web browser and limited programming experience, you can develop and deploy professional applications. We wanted to extend the E-Business Suite functionality with Apex features. Apex functions need to be available from the EBS user menu. Users have to be able to move seamlessly from EBS to Apex, without logging in twice. The context of the EBS user needs to specify which data the user is allowed to see in Apex. Our goal was to develop a calendar in Apex which shows a team’s absences for the logged in supervisor. In EBS the “supervisor -> employee” relation has a hierarchical structure, which has to be available in the calendar.

The first challenge we have to deal with is authentication. Apex users have to log in to access an application. We used the EBS security to allow users to logon to Apex using their EBS username and password. In Apex we created a custom authentication scheme which calls an external function to validate the username and password. The authentication scheme calls our plsql function “apex\_authorise” which uses the EBS “fnd\_web\_sec.validate\_login” to validate the username and password. At this point we are able to log in to an Apex application using any valid EBS username and password combination.

Now we want to link Apex forms from the EBS menu. The first problem is that Apex requires a valid password for the user and the EBS database does not contain passwords, it holds a hash of the username/password combination. You can handle this by creating an alternative password using the DBMS\_OBFUSCATION\_TOOLKIT md5 function. The password will be generated by hashing the username, a time component and a key held within the database. This mechanism will be handled in two functions: “apex\_generate\_hash” and “apex\_validate\_hash”. To check if the password is valid, we extend the already created function “apex\_authorise”.

Now we want to login in Apex automatically. We use a cookie to achieve this. The form function we use in EBS, sets a cookie containing the username and password and redirects to the Apex URL to launch the Apex application. In EBS we have to create a web enabled mod\_plsql procedure to achieve this. This form function can then be added to the EBS menu. In Apex we use a “before header” process to read the cookie and log the user in.

The cookie we used:

OWA\_COOKIE.send  
(name=>’APEX\_APPS\_’||application,  
value=>FND\_GLOBAL.user\_name||’:’||  
apex\_generate\_hash(FND\_GLOBAL.user\_name)||’:’||  
FND\_GLOBAL.user\_id||’:’||  
FND\_GLOBAL.resp\_id||’:’||  
FND\_GLOBAL.resp\_appl\_id  
);

We store the FND\_GLOBAL values in global variables in Apex. We use these variables in the page query of an Apex application to only show the results for this specific user. In our case we used the standard calendar feature of Apex. In this calendar you can specify a sql query which selects the data to publish.

Ideally we want to save the context of the environment an APPS user is working in and seamlessly use this context in our Apex application. This way we don’t have to care about which data is visible for the user, the APPS security mechanism will take care of that. This is something which has to be investigated. In our project we send the user information in the cookie, and use this information for our data selection.

 Viewed 16199 times by 5079 visitors

This is a highly insecure approach - you should look into handling it via cookies.   
  
However, IF you insist on doing logins in this way, you need to read the documentation about the APEX URL structure.  
  
<http://download.oracle.com/docs/cd/E23903_01/doc/doc.41/e21674/concept_url.htm#BEIFCDGF>  
  
So your link would look something like  
  
<http://localhost:8080/apex/f?p=300:1:::::Username,password:test,test>  
  
although, the question still remains how would you get the user's credentials (username & password) in the first place, if the user doesn't type it in?

# Custom Authentication in APEX

Trust is a great force multiplier. – Tom Ridge When you build a web application you have two choices for the pages: they can either be public or protected. Mostly you will build applications using a combination of the two. Some pages will be publicly available, like the landing page of your application, but there will also be pages which

*Trust is a great force multiplier. – Tom Ridge*

When you build a web application you have two choices for the pages: they can either be public or protected. Mostly you will build applications using a combination of the two. Some pages will be publicly available, like the landing page of your application, but there will also be pages which are protected. Users need to login to the application to use these pages.

## Authentication methods

APEX provides us with a couple of authentication schemes by default. Some of which are:

* Application Express – Every user must exist as an APEX user
* Database Account – Every user must have a database account
* Open Door Credentials

The custom authentication is a good option so you can keep full control over how you want this to work. When you have created your application, or at least the start pages, you go to the shared components page where you go select the Authentication Schemes from the Security section.

## Custom authentication scheme

When you want to create your own authentication scheme you must create a (packaged) function that must obey a few rules. The function must accept two parameters: the first is the username, and the second is the password. Both these parameters are varchar2 type. Remember, the username and password parameter are sent as clear text. If you want your application to be more secure, you may want to obfuscate the values before sending them to the authentication function. The simplest form of the authentication function is like this:

FUNCTION authenticate(username\_in IN VARCHAR2

,password\_in IN VARCHAR2)

RETURN BOOLEAN

IS

BEGIN

RETURN TRUE;

END authenticate;

This function is pretty much the same as the open door version, but it’s the start of a real authentication. You create a new authentication scheme by selecting the create button on the screen with the list of defined schemes for this application.

When you create a new scheme you can choose to create a scheme based on an existing scheme, but in this case we want to create a new scheme based on one of the pre-configured schemes. In this screen you can create the code for the Authentication Function. You can write the code for the function in this screen, but you can also create the function as a stored (package) function so you can use your IDE to create the code.

To create a real authentication scheme you need to do more than just return true for whatever parameters are sent in. You can write code to check the usernames and their passwords, but that would mean you would have to alter the code every time you want to add or remove a user. You are better off using a table which stores the users for the application. The table can look like this: Table USERS

|  |  |  |
| --- | --- | --- |
| **ID** | NUMBER(15,0) | Primary key |
| **USERNAME** | VARCHAR2(50) | Unique constraint |
| **PASSWORD** | VARCHAR2(50) |  |

You can this table to save more information on the user, such as the email address but for this example that is not necessary. The function could be updated to something like this:

FUNCTION authenticate(username\_in IN VARCHAR2

,password\_in IN VARCHAR2) RETURN BOOLEAN IS

l\_value NUMBER;

l\_returnvalue BOOLEAN;

BEGIN

BEGIN

SELECT 1

INTO l\_value

FROM users

WHERE 1 = 1

AND upper(users.username) = upper(username\_in)

AND upper(users.password) = upper(password\_in);

EXCEPTION

WHEN no\_data\_found

OR too\_many\_rows THEN

l\_value := 0;

WHEN OTHERS THEN

l\_value := 0;

END;

l\_returnvalue := l\_value = 1;

RETURN l\_returnvalue;

END;

In this example the username and password are validated against the data in the table. The username and password are stored in plain text in the database, which is not a good idea. You may want to obfuscate the password before storing it. This is where creating a package for the authentication functions comes in handy. You want to use the same function for obfuscating the password when you store the data as when you check the credentials. You can do this by creating a private function in the package that does the obfuscating and use the outcome of this function for both storing the data and checking the entered credentials. A package like this could look like this:

PACKAGE redgate\_authentication IS

PROCEDURE adduser(username\_in IN VARCHAR2

,password\_in IN VARCHAR2);

FUNCTION authenticate(username\_in IN VARCHAR2

,password\_in IN VARCHAR2) RETURN BOOLEAN;

END redgate\_authentication;

PACKAGE BODY redgate\_authentication IS

-- private functions

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\ || function : obfuscate || parameters : text\_in -=> text to be obfuscated || || return value: obfuscated value || || purpose : Hash the value of text\_in || || author : PBA || (C) 2013 : Patrick Barel \\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

FUNCTION obfuscate(text\_in IN VARCHAR2) RETURN RAW IS

l\_returnvalue RAW(16);

BEGIN

dbms\_obfuscation\_toolkit.md5(input => utl\_raw.cast\_to\_raw(text\_in), checksum => l\_returnvalue);

RETURN l\_returnvalue;

END obfuscate;

-- public functions

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\ || procedure : adduser || parameters : username\_in -=> Username of the user to be authenticated || password\_in -=> Password of the user to be authenticated || || purpose : Add a user to the users table || || author : PBA || (C) 2013 : Patrick Barel \\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PROCEDURE adduser(username\_in IN VARCHAR2

,password\_in IN VARCHAR2) IS

l\_obfuscated\_password users.password%TYPE;

BEGIN

l\_obfuscated\_password := obfuscate(text\_in => password\_in);

INSERT INTO users

(id

,username

,password)

VALUES

(users\_seq.nextval

,username\_in

,l\_obfuscated\_password);

NULL;

END adduser;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\ || function : authenticate || parameters : username\_in -=> Username of the user to be authenticated || password\_in -=> Password of the user to be authenticated || || return value: TRUE -=> User is authenticated || FALSE -=> User is not authenticated || || purpose : Check if a user is authenticated based on the username and || password supplied || || author : PBA || (C) 2013 : Patrick Barel \\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

FUNCTION authenticate(username\_in IN VARCHAR2

,password\_in IN VARCHAR2) RETURN BOOLEAN IS

l\_obfuscated\_password users.password%TYPE;

l\_value NUMBER;

l\_returnvalue BOOLEAN;

BEGIN

l\_obfuscated\_password := obfuscate(text\_in => password\_in);

BEGIN

SELECT 1

INTO l\_value

FROM users

WHERE 1 = 1

AND upper(users.username) = upper(username\_in)

AND users.password = l\_obfuscated\_password;

EXCEPTION

WHEN no\_data\_found

OR too\_many\_rows THEN

l\_value := 0;

WHEN OTHERS THEN

l\_value := 0;

END;

l\_returnvalue := l\_value = 1;

RETURN l\_returnvalue;

END authenticate;

END redgate\_authentication;

Now all you have to do is tell your application that it needs to reference the custom authentication schema. When you created the new authentication schema your application was automatically told to use the new schema, but if you created an authentication scheme in one application and you created a new application where you copied the authentication scheme from an existing application you have to do this yourself. It can also happen that during development you want to use a different authentication scheme than you might in production.

To change the current authentication scheme you go to the Shared components – Authentication Schemes. Select the scheme you want to use and press the ‘Make Current Scheme’ button.

If the scheme you selected is already the current scheme, then this button will not be available.

## Subscribe for more articles

Fortnightly newsletters help sharpen your skills and keep you ahead, with articles, ebooks and opinion to keep you informed.

Top of Form

Email

Bottom of Form