declare

c\_workspace constant apex\_workspaces.workspace%type := 'DEMO';

c\_app\_id constant apex\_applications.application\_id%type := 20000;

c\_app\_alias constant apex\_applications.alias%type := 'MYDEMO';

l\_workspace\_id apex\_workspaces.workspace\_id%type;

begin

apex\_application\_install.clear\_all;

select workspace\_id

into l\_workspace\_id

from apex\_workspaces

where workspace = c\_workspace;

apex\_application\_install.set\_workspace\_id(l\_workspace\_id);

apex\_application\_install.set\_application\_id(c\_app\_id);

apex\_application\_install.set\_application\_alias(c\_app\_alias);

apex\_application\_install.generate\_offset;

end;

/

@f10000.sql

================Exporting Apex Application Files(Linux Environment) ===========

1 . Set environment variables to execute java APEXExport utility

export JAVA\_HOME=/usr/lib/jvm/jre-1.7.0-openjdk.x86\_64/bin

export  ORACLE\_HOME=/u01/app/oracle/product/11.2.0/dbhome\_1

export CLASSPATH=.:${ORACLE\_HOME}/jdbc/lib/ojdbc6.jar

2. Go to apex utilities directory

/u01/app/oracle/product/11.2.0/dbhome\_1/apex/utilities

3. Execute below commands

**For Single Application export -----**

java oracle.apex.APEXExport -db <db\_server\_IP>:<port\_no>:<sid>  -user <schema\_name> -password <password> -applicationid <application\_id>

Example:

java oracle.apex.APEXExport -db 10.40.10.100:1521/service\_name -user scott -password scott -applicationid 603

\*\*it will generate f603.sql application files for every application inside the workspace

\*\*You need to use ‘/’ in place of ‘:’ while using service name instead of SID.

\*\*You can export remote server apex applications using the db server IP.

**For Full Workspace Applications export -----**

java oracle.apex.APEXExport -db <db\_server\_IP>:<portno>:<sid>  -user <schema\_name> -password <password> -workspaceid <workspace\_id>

Example :

java oracle.apex.APEXExport -db 10.40.10.100:1521:ORCL -user scott -password scott -workspaceid 4651

\*\*it will generate .sql application files for every application inside the workspace

================Importing Apex Application Files(Linux Environment) ===========

1 . Move the .sql files to the server where you wanted to import the applications.

2. Set environment variables to connect to database

export ORACLE\_HOME=/u01/app/oracle/product/11.2.0/dbhome\_1

export ORACLE\_SID=orcl

export PATH=$PATH:$ORACLE\_HOME/bin

export ORACLE\_BASE=/u01/app/oracle

2. Write shell script to import all applications in one time / Execute .sql by logging to db

The is the content of shell script to import multiple .sql application files. You need to login to the shema where applications need to be imported.

sqlplus -s "scott/scott" << EOF >>import.log

@f180.sql;

@f181.sql;

@f231.sql;

exit ;

EOF

[**app\_export.sh**](https://gist.github.com/aimtiaz11/32b91c0e2ffb49c4786b#file-app_export-sh)

|  |  |
| --- | --- |
|  | #!/bin/bash |
|  | # To set password value |
|  | cd ~/ && . .bashrc |
|  |  |
|  | DB\_HOSTNAME=localhost # Required Info |
|  | export DB\_HOSTNAME |
|  |  |
|  | DB\_PORT=1521 # Required Info |
|  | export DB\_PORT |
|  |  |
|  | DB\_SERVICE\_ID=XE # Required Info |
|  | export DB\_SERVICE\_ID |
|  |  |
|  | APP\_ID=100 # Required Info |
|  | export APP\_ID |
|  |  |
|  | APP\_PARSING\_SCHEMA=MY\_SCHEMA # Required Info |
|  | export APP\_PARSING\_SCHEMA |
|  |  |
|  | BACKUP\_DEST=/home/oracle/apex\_application\_backups # Required Info |
|  | export BACKUP\_DEST |
|  |  |
|  |  |
|  | # Apex Install directory |
|  | APEX\_HOME=/home/oracle/staging/apex # Required Info |
|  | export APEX\_HOME |
|  |  |
|  | ORACLE\_HOME=/u01/app/oracle/product/11.2.0/xe # Required Info |
|  | export ORACLE\_HOME |
|  |  |
|  | CLASSPATH=$ORACLE\_HOME/jdbc/lib/ojdbc6.jar:./ # Required Info |
|  | export CLASSPATH |
|  |  |
|  | # Export run from here |
|  | cd $APEX\_HOME/utilities |
|  |  |
|  | echo Running Export....... |
|  |  |
|  | # run the export...DB\_PASS from .bashrc |
|  | /home/oracle/jdk1.7.0\_51/bin/java oracle.apex.APEXExport -db $DB\_HOSTNAME:$DB\_PORT:$DB\_SERVICE\_ID -user $APP\_PARSING\_SCHEMA -password $DB\_PASS -expPubReports -expSavedReports -expIRNotif -applicationid $APP\_ID |
|  |  |
|  | # Copy to backup Dest |
|  | cp f100.sql $BACKUP\_DEST |
|  |  |
|  | echo .....Removing the export from $APEX\_HOME/apex/utilities after copy |
|  | rm f100.sql |
|  | echo removed |
|  |  |
|  | cd $BACKUP\_DEST |
|  |  |
|  | \_now=$(date +"%Y%m%d") |
|  |  |
|  | /usr/local/bin/git add \* |
|  | /usr/local/bin/git commit -m "Scheduled backup via crontab "$\_now |
|  | /usr/local/bin/git push origin master |

[**app\_import.sql**](https://gist.github.com/aimtiaz11/32b91c0e2ffb49c4786b#file-app_import-sql)

|  |  |
| --- | --- |
|  | set serveroutput on |
|  | WHENEVER SQLERROR EXIT SQL.SQLCODE |
|  | declare |
|  | l\_workspace\_id number; |
|  |  |
|  | -- command line configuration |
|  | -- \*\* MODIFY WITH CAUTION - CAN BREAK/OVERWRITE OTHER APEX APPLICATIONS IF CONFIGURED WRONG \*\* |
|  | l\_workspace\_name varchar2(100) := 'GEOCODER'; |
|  | l\_application\_id number := 502; |
|  | l\_parsing\_schema varchar2(100) := 'GEOCODER'; |
|  | begin |
|  | select workspace\_id into l\_workspace\_id |
|  | from apex\_workspaces |
|  | where upper(workspace) = upper(l\_workspace\_name); |
|  | -- |
|  | apex\_application\_install.set\_workspace\_id( l\_workspace\_id ); |
|  | apex\_application\_install.set\_application\_id(l\_application\_id); |
|  | apex\_application\_install.generate\_offset; |
|  | apex\_application\_install.set\_schema( l\_parsing\_schema ); |
|  | apex\_application\_install.set\_application\_alias( 'F' || apex\_application\_install.get\_application\_id ); |
|  | end; |
|  |  |
|  | / |
|  | @f100.sql |

[**GitInstall**](https://gist.github.com/aimtiaz11/32b91c0e2ffb49c4786b#file-gitinstall)

|  |  |
| --- | --- |
|  | # yum install curl-devel expat-devel gettext-devel \ openssl-devel zlib-devel |
|  |  |
|  | # wget https://git-core.googlecode.com/files/git-1.9.0.tar.gz |
|  | # tar -xvzf git-1.9.0.tar.gz |
|  | # cd git-1.9.0 |
|  |  |
|  | # make prefix=/usr/local all |
|  | # make prefix=/usr/local install |

**Import Apex Application at command line**

Recently (since 4.1 update) I’ve experienced issues when importing an application into an Apex environment.

After a little bit of searching I found the way to do the import on the command-line.

It is documented in the Apex documentation, but as you know I’d like this blog to be an “OPI” (one point of information)

Please have a look at the script, alter it whenever appropriate.

declare

-- the name of the workspace in wich to import

t\_workspace varchar2(30):= 'MERIDA';

-- an application number of an existing application in the workspace.

t\_existing\_app number := 100;

-- the "new" application number, an existing number will be dropped first.

t\_new\_app number := 100;

-- security group id, you don't have to set this variable

t\_secgrp\_id number;

begin

-- get the Security Group ID

select workspace\_id

into t\_secgrp\_id

from apex\_applications

where application\_id = t\_existing\_app;

-- Set the Security Group ID

wwv\_flow\_api.set\_security\_group\_id(p\_security\_group\_id => t\_secgrp\_id);

-- Set the Application ID to use

apex\_application\_install.set\_application\_id(t\_new\_app);

-- This procedure generates the offset value used during application import.

-- The offset value is used to ensure that the metadata for the Application

-- Express application definition does not collide with other metadata on

-- the instance. For a new application installation, it is usually sufficient

-- to call this procedure to have Application Express generate this offset

-- value for you.

apex\_application\_install.generate\_offset;

-- Set the parsing schema

apex\_application\_install.set\_schema( t\_workspace );

-- Set the application alias

apex\_application\_install.set\_application\_alias( 'F' || t\_new\_app );

end;

/

-- Do the actual import

@D:ProjectsStaplesMeridaexportsmerida-srv022-20111222-1157.sql;

commit;

**Deploying Application Express on the Command Line**

[](https://jeffkemponoracle.com/wp-content/uploads/2013/05/export.gif)I love the APEX UI, it makes development so much easier and more convenient – and makes it easy to impress clients when I can quickly fix issues right there and then, using nothing but their computer and their browser, no additional software tools needed.

However, at my main client they have a fairly strict “scripted releases only” policy which is a really excellent policy – deployments must always be provided as a script to run on the command line. This makes for less errors and a little less work for the person who runs the deployment.

In APEX it’s easy to create deployment scripts that will run right in SQL\*Plus. You can export a workspace, an application, images, etc. as scripts that will run in SQL\*Plus with *almost* no problem. There’s just a few little things to be aware of, and that’s the subject of this post.

**1. Setting the session workspace**

Normally if you log into APEX and import an application export script, it will be imported without problem. Also, if you log into SQL\*Plus and try to run it, it will work fine as well.

The only difference comes if you want to deploy it into a different workspace ID to the one the application was exported from – e.g. if you want to have two workspaces on one database, one for dev, one for test, when you log into your test schema and try to run it, you’ll see something like this:

[?](https://jeffkemponoracle.com/2013/05/deploying-application-express-on-the-command-line/)

|  |
| --- |
| SQL&gt; @f118.sql  APPLICATION 118 - My Wonderful App  Set Credentials...  Check Compatibility...  Set Application ID...  begin  \*  ERROR at line 1:  ORA-20001: Package variable g\_security\_group\_id must be set.  ORA-06512: at &quot;APEX\_040100.WWV\_FLOW\_API&quot;, line 73  ORA-06512: at &quot;APEX\_040100.WWV\_FLOW\_API&quot;, line 342  ORA-06512: at line 4 |

*Side note: if you’re using Windows, the SQL\*Plus window will disappear too quickly for you to see the error (as the generated apex script sets it to exit on error) – so you should SPOOL to a log file to see the output.*

To fix this issue, you need to run a little bit of PL/SQL before you run the export, to override the workspace ID that the script should use:

[?](https://jeffkemponoracle.com/2013/05/deploying-application-express-on-the-command-line/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | declare    v\_workspace\_id NUMBER;  begin    select workspace\_id into v\_workspace\_id    from apex\_workspaces where workspace = 'TESTWORKSPACE';    apex\_application\_install.set\_workspace\_id (v\_workspace\_id);    apex\_util.set\_security\_group\_id      (p\_security\_group\_id =&gt; apex\_application\_install.get\_workspace\_id);    apex\_application\_install.set\_schema('TESTSCHEMA');    apex\_application\_install.set\_application\_id(119);    apex\_application\_install.generate\_offset;    apex\_application\_install.set\_application\_alias('TESTAPP');  end;   / |

This will tell the APEX installer to use a different workspace – and a different schema, application ID and alias as well, since 118 already exists on this server. If your app doesn’t have an alias you can omit that last step. Since we’re changing the application ID, we need to get all the other IDs (e.g. item and button internal IDs) throughout the application changed as well, so we call generate\_offset which makes sure they won’t conflict.

**2. Installing Images**

This is easy. Same remarks apply as above if you’re installing the image script into a different workspace.

**3. Installing CSS Files**

If you export your CSS files using the APEX export facility, these will work just as well as the above, and the same considerations apply if you’re installing into a different workspace.

If you created your CSS export file manually using *Shared Components* -> *Cascading Style Sheets* and clicking on your stylesheet and clicking “*Display Create File Script*“, you will find it doesn’t quite work as well as you might expect. It does work, except that the file doesn’t include a COMMIT at the end. Which normally wouldn’t be much of a problem, until you discover late that the person deploying your scripts didn’t know they should issue a commit (which, of course, would have merely meant the file wasn’t imported) – and they didn’t actually close their session straight away either, but just left it open on their desktop while they went to lunch or a meeting or something.

This meant that when I sent the test team onto the system, the application looked a little “strange”, and all the text was black instead of the pretty colours they’d asked for – because the CSS file wasn’t found. And when I tried to fix this by attempting to re-import the CSS, my session hung (should that be “hanged”? or “became hung”?) – because the deployment person’s session was still holding the relevant locks. Eventually they committed their session and closed it, and the autocommit nature of SQL\*Plus ended up fixing the issue magically for us anyway. Which made things interesting the next day as I was trying to work out what had gone wrong, when the system was now working fine, as if innocently saying to me, “what problem?”.

**4. A little bug with Data Load tables**

We’re on APEX 4.1.1  If you have any CSV Import function in your application using APEX’s Data Loading feature, if you export the application from one schema and import into another schema, you’ll find that the Data Load will simply not work, because the export incorrectly hardcodes the owner of the data load table in the call to create\_load\_table. This bug is described here: <http://community.oracle.com/message/10309103?#10307103> and apparently there’s a patch for it.

[?](https://jeffkemponoracle.com/2013/05/deploying-application-express-on-the-command-line/)

|  |
| --- |
| wwv\_flow\_api.create\_load\_table(   p\_id =&gt;4846012021772170+ wwv\_flow\_api.g\_id\_offset,   p\_flow\_id =&gt; wwv\_flow.g\_flow\_id,   p\_name =&gt;'IMPORT\_TABLE',   p\_owner =&gt;'MYSCHEMA',   p\_table\_name =&gt;'IMPORT\_TABLE',   p\_unique\_column\_1 =&gt;'ID',   p\_is\_uk1\_case\_sensitive =&gt;'Y',   p\_unique\_column\_2 =&gt;'',   p\_is\_uk2\_case\_sensitive =&gt;'N',   p\_unique\_column\_3 =&gt;'',   p\_is\_uk3\_case\_sensitive =&gt;'N',   p\_wizard\_page\_ids =&gt;'',   p\_comments =&gt;''); |

The workaround I’ve been using is, before importing into a different schema, I just edit the application script to fix the p\_owner in the calls to wwv\_flow\_api.create\_load\_table.

**5. Automating the Export**

I don’t know if this is improved in later versions, but at the moment you can only export Applications using the provided API – no other objects (such as images or CSS files). Just a sample bit of code (you’ll need to put other bits around this to do what you want with the clob – e.g. my script spits it out to serverout so that SQL\*Plus will write it to a sql file):

[?](https://jeffkemponoracle.com/2013/05/deploying-application-express-on-the-command-line/)

|  |
| --- |
| l\_clob := WWV\_FLOW\_UTILITIES.export\_application\_to\_clob    (p\_application\_id =&gt; &amp;APP\_ID.    ,p\_export\_ir\_public\_reports =&gt; 'Y'    ,p\_export\_ir\_private\_reports =&gt; 'Y'    ,p\_export\_ir\_notifications =&gt; 'Y'    ); |

That’s all my tips for scripting APEX deployments for now. If I encounter any more I’ll add them here.

**EDIT:**

Related: “What’s the Difference” – comparing exports to find diffs on an APEX application – http://blog.sydoracle.com/2011/11/whats-difference.html