How much time does it take for your team to deliver your freshly updated applications to all the environments and customers? In case you just play with APEX and ORDS, it is totally ok for you to export and import your workspaces, applications and RESTful Services definitions *manually* using web-based interface (under *App Builder* > *Export/Import* and *SQL Workshop* > *RESTful Services* > *RESTful Data Services*). But it is obviously not a suitable way for more or less serious development, and, surely, for the enterprise use.

**Why so?**

Let's imagine you have a team of several developers who do different job and once a couple of days you need to deliver their changes to all the environments. If your team develops a single application in a single workspace, which is supposed to be deployed to a **single test** environment and a **single production** server, you still can do all this *manually*. It's just not the time for you. Yet.

But imagine you have several testing platforms, and dozens or even hundreds of customers. In such a case it becomes more or less *impossible* to deliver all the changes as frequently as you could do, providing you had the only customer. However, many people continue exploiting the *manual* approach, and don't even realise they're doing it wrong.

That's why we need automation - to stop [losing our time](https://oracle-base.com/blog/2018/06/12/why-automation-matters-lost-time/). More than two years ago [*Martin D'Souza*](https://twitter.com/martindsouza) [already wrote](https://insum.ca/oracle-apex-deployments-youre-doing-it-wrong/) about how APEX applications could be exported and imported using *SQLcl*. That article was a good start for me, though, I didn't want to be dependant on SQLcl or any other specific tool, and needed to do even more - deploy not only *APEX applications*, but also *APEX workspaces* and *RESTful Data Services modules definitions*.

So, my ultimate goal was to work out a way to create an applicable set of SQL-scripts with the definitions of all the mentioned database objects from the database metadata.

Additionally, this set of scripts had to be *universal*, which means they had to be compatible with the *SQL\*Plus*, *SQLcl* and *SQL Developer*.

And this is what I finally ended up with.

**The Export**

**APEX Workspace**

The first thing you deploy in order for your APEX applications to work, is your workspaces. Actually, if you have the only server in production environment, automation of this step is quite *optional* and you can still do the job using the stanard web-based interface (*Instance Administration > Manage Workspaces > Export Import > Export Workspace*). But if you're going to install your applications on some number of hosts with their own database instances, it's a very good idea not to do everything by your bare hands.

To export workspace definitions we are going to use [APEX\_EXPORT.GET\_WORKSPACE](https://docs.oracle.com/database/apex-18.2/AEAPI/GET_WORKSPACE_Function.htm#AEAPI-GUID-493DB29D-8740-418E-9AF9-419EDE564CC5) API. Mind the fact that you need to run the script below as a parsing schema user (which is associated with the workspace), or as SYS, otherwise you're going to get the ORA-20987: APEX - Security Group ID (your workspace identity) is invalid. - Contact your application administrator exception.

-- connecting to the instance (parsing schema user could be used instead of SYS)

connect sys as sysdba

set timing on

timing start TIMER\_WORKSPACE\_EXPORT

-- SQL\*Plus environment settings

-- they are crucial to retrieve a consistent export file

set feedback off

set heading off

set echo off

set flush off

set termout off

set pagesize 0

set long 100000000 longchunksize 32767

column output format a4000

set linesize 4000

set trimspool on

-- variable for storing the export data

variable contents clob

-- generating the export file

declare

l\_files apex\_t\_export\_files;

begin

l\_files := apex\_export.get\_workspace(

p\_workspace\_id => 100500,

p\_with\_date => false,

p\_with\_team\_development => false,

p\_with\_misc => true

);

-- in the first file we have the needed export data

:contents := l\_files(1).contents;

end;

/

-- you can specify any file name here

spool my\_workspace\_export.sql

print contents

spool off

timing stop TIMER\_WORKSPACE\_EXPORT

exit

Notice that the size of the export data can be large (though this is mostly true for application exports), that's why we need to set proper SQL\*Plus environment settings.

**APEX Application**

Exporting of your APEX applications looks very similar to what you've just seen above. But there's a huge difference in the frequency of its usage - you need to export an application every time you need to get a new version of it in order to deliver the changes to your customers. Even if there's only one.

As before, to export application we are going to use APEX's PL/SQL API, but this time it's going to be [APEX\_EXPORT.GET\_APPLICATION](https://docs.oracle.com/database/apex-18.2/AEAPI/GET_APPLICATION_Function.htm#AEAPI-GUID-A8E626D6-D7DE-4E59-8F90-3666A7A41A87). Again, you must connect as a parsing schema user (which is associated with the workspace with the application to export), or as SYS to run the script.

-- connecting to the instance (parsing schema user could be used instead of SYS)

connect sys as sysdba

set timing on

timing start TIMER\_APP\_EXPORT

-- SQL\*Plus environment settings

-- they are crucial to retrieve a consistent export file

set feedback off

set heading off

set echo off

set flush off

set termout off

set pagesize 0

set long 100000000 longchunksize 32767

column output format a4000

set linesize 4000

set trimspool on

-- variable for storing the export data

variable contents clob

-- generating the export file

declare

l\_files apex\_t\_export\_files;

begin

l\_files := apex\_export.get\_application(

p\_application\_id => 150,

p\_with\_ir\_public\_reports => true,

p\_with\_ir\_private\_reports => true,

p\_with\_ir\_notifications => true,

p\_with\_translations => true,

p\_with\_comments => true,

p\_with\_acl\_assignments => true

);

:contents := l\_files(1).contents;

end;

/

-- you can specify any file name here

spool my\_app\_export.sql

print contents

spool off

timing stop TIMER\_APP\_EXPORT

exit

As a result you will get a huge SQL-script which contains metadata of all the application pages and even static application files. You also have an option to generate a separate file for each application page if you specify the p\_split parameter value as true.

**ORDS Modules**

In this chapter we are going to consider only the ORDS-based RESTful Services, which is quite a standard at the moment.

To export ORDS modules you, as usual, can use the comfortable APEX's browser interface (under *SQL Workshop > RESTful Services > RESTful Data Services > Export*), but again it doesn't hurt much to automate the process, at least if your RESTful services tend to evolve through time. And once more time, there's an existing [blog post](https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-use-sqlcl-to-display-ords-web-service-definitions) by [Tim Hall](https://twitter.com/oraclebase) on how to do all this using *SQLcl*. And still, I didn't want to be dependant on it, hence needed a different approach.

So, we're going to use the ORDS\_EXPORT PL/SQL package in the ORDS\_METADATA schema, which is, unfortunately, still not documented properly. Actually, under the hood exactly this package is used by the APEX SQL Workshop's pages.

To get a correct export file, you need to run the script below as a REST-enabled schema user.

-- connecting to the instance (you must use the REST-enabled schema user - RESTAPI in this case)

connect restapi

set timing on

timing start TIMER\_REST\_EXPORT

-- SQL\*Plus environment settings

-- they are crucial to retrieve a consistent export file

set feedback off

set heading off

set echo off

set flush off

set termout off

set pagesize 0

set long 100000000 longchunksize 32767

column output format a4000

set linesize 4000

set trimspool on

-- variable for storing the export data

variable contents clob

-- generating the export file

begin

:contents := ords\_metadata.ords\_export.export\_schema();

end;

/

-- you can specify any file name here

spool my\_rest\_modules\_export.sql

print contents

-- the trailing '/' symbol is needed to terminate the generated PL/SQL block

prompt /

spool off

timing stop TIMER\_REST\_EXPORT

exit

After running the script, you end up with exactly the same SQL-script as though you used the SQL Workshop's interface.

It's also possible to export a particular module, if you like, for this exact purpose there's another function export\_module(p\_module\_name => 'MY\_MODULE') in the ORDS\_EXPORT package. Check out the package specification for more info on this.

**The Import**

Generally, the import process is quite simple if you did everything right, since all the exported files are usual SQL-scripts which can be executed by your favourite tool. The only difference here is the user as whom you should run the scripts.

**APEX Workspace**

To import a workspace, you must be connected to the database as APEX internal user (for instance, APEX\_180100) or as SYS (but in this case you should change your current schema to the APEX internal user's by invoking, in our case, alter session set current\_schema = APEX\_1801000; statement).

Note the fact that if you go for the APEX internal user, it is usually in the *locked* state, so you first need to unlock it and set a password for the user. After doing the job (the import), the APEX internal user must be locked back, because leaving it unlocked is [way dangerous](https://dsavenko.me/how-apex-executes-sql-statements-without-having-permissions/).

You also should be in the directory where the script resides, since we're going to load it directly.

-- connecting to the instance as SYS

connect sys as sysdba

-- changing the current schema

alter session set current\_schema = APEX\_180100;

set echo on

set timing on

set serveroutput on

set linesize 120

whenever sqlerror continue none

timing start TIMER\_WORKSPACE\_IMPORT

spool workspace\_import.log

@my\_workspace\_export.sql

spool off

set serveroutput off

purge recyclebin;

timing stop TIMER\_WORKSPACE\_IMPORT

exit

**APEX Application**

Things could be a little different with the APEX applications import, due to the fact it has its own API to configure the process. It's done by using the APEX\_APPLICATION\_INSTALL package. Though, its usage is totally optional and is needed only if you intend to change something about the application before the actual import. There's even a [set of examples](https://docs.oracle.com/database/apex-18.2/AEAPI/Import-Script-Examples.htm#AEAPI560) on how to use the package in the official documentation.

Here I'm going to show the trivial situation when you're okay with the defaults (no need to change of application ID, offset, workspace, alias or any other application property). The script below should be run as a parsing schema user.

-- connecting to the instance as a parsing schema user APEX\_APP

connect apex\_app

set echo on

set timing on

set serveroutput on

set linesize 120

whenever sqlerror continue none

timing start TIMER\_APP\_IMPORT

spool app\_import.log

@my\_app\_export.sql

spool off

set serveroutput off

purge recyclebin;

timing stop TIMER\_APP\_IMPORT

exit

**ORDS Modules**

To import your recently exported RESTful Data Services definitions you should switch to the REST-enabled schema user and change the name of the script.

-- connecting to the instance as a REST-enabled schema user RESTAPI

connect restapi

set echo on

set timing on

set serveroutput on

set linesize 120

whenever sqlerror continue none

timing start TIMER\_REST\_IMPORT

spool rest\_services\_import.log

@my\_rest\_modules\_export.sql

spool off

set serveroutput off

purge recyclebin;

timing stop TIMER\_REST\_IMPORT

exit

That's it!

Obviously, the mentioned scripts could be put together when you work with your particular situation and when you know what you're doing. The only point here is that separate scripts give you more freedom when it comes to automation. You can set up your own pipelines to run particular scripts when a particular event occurs.

**But where is the automation?**

Right. We only wrote a bunch of scripts and nothing more. But as I like to say, *automation starts where interactivity stops*. All web-based user interfaces are interactive and need a human to function, but the set of scripts, which we worked out, could be used in both interactive and non-interactive ways. Which means, you can definitely use them with your favourite CI/CD tool.

It could cost you some time and effort in the beginning to set up all this, but will save much more in the future. And stopping wasting time on the routine on every day basis is not the only reason. Automation of deployments enables you to implement such things as automated testing, continious integration and delivery, which, in their turn, decrease number of bugs in your code and shortens the time your customers see the brand-new funtionality.

I think it's enough to try, isn't it?

Do you have any other ideas how to automate development process with APEX and ORDS worth sharing? You're warmly welcome to put it here in the comments section - we'll definitely discuss them all!