ORACLE

APPLICATION EXPRESS

An Oracle White Paper
September 2016

Life Cycle Management with Oracle Application Express (Revision 2)

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Table of Contents

Executive Overview

Oracle Application Express (APEX) is a web-based development and deployment tool that is available with all Oracle databases and in all Oracle Database Cloud Services. It is used to create modern database-centric web applications that are reliable, scalable, and secure. These applications can range from small departmental applications built by one or two people, to large, complex, business critical systems developed by a team of IT professionals.

This paper provides specific recommendations to optimize the development and deployment of Oracle Application Express applications throughout their life-cycle, including the use of 3rd party source control products for maintaining version control. The governance of any large scale development requires an effective framework to enable multiple developers to work concurrently, and the ability to manage the deployment of multiple versions of the application from Development, through QA/Test, and into Production.

Application Express is designed to allow any number of developers to work on the same applications concurrently. However, unlike file based languages, the application definitions are stored as meta-data inside the Oracle Database. Rather than checking-out individual artifacts, developers work within the Application Builder, to review and maintain the application definitions. In the development environment, they can simply open an application, navigate to a specific page, and then start making enhancements. This flexibility has proven to be a highly productive manner to very rapidly build and enhance applications. The Application Builder, that allows multiple developers to work on the same application at the same time, requires different development techniques, rather than simply checking-out / checking-in of artifacts employed with file-based development.

Oracle recommends implementing the majority of your business logic in PL/SQL, rather than within the application definitions. These packages, functions, procedures, and so forth, should be maintained in external script files outside of the Oracle database, so that they can easily be maintained in source control.

For proof that maintaining version control with large development teams and complex applications is possible with Oracle Application Express, look no further than the Application Express Engine itself. The pages within the Application Express Engine are built with Application Express. The engine consists of several very large and complex applications, consisting of hundreds of pages, together with over 300,000 lines of PL/SQL programs, plus numerous CSS, JavaScript, HTML, and static files.

Generally, standard best practices for developing software applications in a full development life cycle should be followed. These include:

- Proper project management to keep to specified timelines and prevent scope creep;
- Good database design;
- Defining and following good application design standards;
- Appropriate functional, security, and performance testing and reviews;
- Proper utilization of a source control repository for all deliverables; and
- Well defined roll-out procedures.

Oracle Application Express Application Components

Application Express is installed into an Oracle Database. For example, when installing Application Express Release 5.0, a schema (APEX_050000) is created inside the Oracle Database. This schema holds the Application Express Engine which consists of hundreds of tables and thousands of lines of PL/SQL. The tables hold the meta-data for all of the applications defined within that installation. Therefore, an application which consists of numerous pages, regions, items, buttons, and shared components, is defined by records in the meta-data tables within the APEX engine schema. For example, when a developer adds an item onto a page, the item definition is transformed into a record which is inserted into one of the meta-data tables in the Oracle database. This same table also holds all of the other items defined across all of the other applications.

Oracle Database Scripts and External Files

An application may reference several different types of database objects, such as tables, views, sequences, database packages, functions, procedures, and so forth. All database objects should be created and maintained using SQL script files. SQL script files should be written to manipulate data, such as adding records to a new base table.

Scripts for creating or updating database objects, manipulating data, and other external components should be checked into version control separately from checking in Application Express applications. These files should be managed as you would for any other file based components. For example, to update a PL/SQL function, the developer should check out the SQL script file from version control and then update the code in the file. Once the code changes are complete, the script should be run in the Development environment, and then checked back into source control.

An application may also reference other external components, such as JavaScript, CSS, HTML, and other static files. These components should be managed as separate files in the source control system.

Supporting Objects

Within an application, developers can define Supporting Objects. Supporting Objects are used to define database object installation and upgrade scripts that are invoked when importing an application. Supporting objects allow you to package both the application and database objects needed in a single file. Supporting Objects are primarily designed for Packaged Apps and commercial applications such that installing an application also installs / updates the database objects. For **in-house** development, Oracle does **not** recommend utilizing Supporting Objects, as such scripts should be maintained outside of the application, so that they can be maintained in external script files, and checked into source control.

Configuring Application Express Environments

When developing with Oracle Application Express you should follow standard system development lifecycle practices, such as having different environments for Development, QA/Test and Production. These environments generally reside in separate databases, or in separate pluggable databases (PDBs) in an Oracle Database 12c Multitenant environment. These environments can either be on premise or use hosted services, such as the Oracle Database Cloud Service, or a combination of both.

Application Express exports can be imported into any other Oracle Database, providing Application Express (that release or higher) is installed into the target database. This provides a lot of flexibility with where development is performed, and where applications are deployed. For example, you may choose to perform development and testing in the Oracle Database Cloud Service, but deploy the production applications on premise, or vice versa. Given that Application Express is identical in all environments, the physical location of your Application Express environments does not impact the recommendations made in this document.

Controlling Application Changes

Developers should only be allowed to make changes to applications and related database objects in the development environment. To further enforce this policy, it is recommended that you install "Runtime Only" Application Express in the QA/Test and Production environments. This will prohibit developers from accessing the Application Builder and SQL Workshop in these environments. This recommendation also helps to ensure that the applications built into these environments are obtained directly from the source code repository. If a developer is able to alter an application in QA/Test or Production, then maintaining version control can become very problematic, and there are no guarantees that the same change will also be made in Development. Therefore, subsequent builds from Development may then overwrite those ad-hoc changes leading to application errors, code loss, and code instability.

Database Administrators (DBAs) should be the only ones with permission to update the QA/Test and Production environments. All application upgrades and environment preferences are performed using the published APIs, instead of developers directly logging into that environment and performing changes.

Application Express Workspaces

A single installation of Application Express can include any number of workspaces. A workspace is a logical store for groups of applications. Different workspaces may be defined for different departments, or functional groups, within an organization. For example, Oracle hosts an internal service (http://apex.oraclecorp.com), which hosts over 3,000 workspaces, and is used by all lines of business within Oracle.

Defining QA/Test and Production Workspaces

To provide maximum flexibility in deploying applications to QA/Test and Production environments it is best to export the workspace from Development and then import that workspace into the other environments. Any Application Express accounts defined in Development should be removed from these environments. In this manner, it will be possible to export application components from Development, such as a single page, and import them into the other environments, providing the original import uses the same Application ID as defined in Development.

Propagating Releases

Once development of a set of features is complete, the developers need to ensure all of the deliverables are checked into a release folder within source control. The complete build includes:

- Application Express application export file(s);
- Any database definition language (DDL) scripts;
- Any data manipulation language (DML) scripts; and
- Any other external files.

The developer responsible for the release should then produce a single *release.sql* script which will run all of the necessary files, and check that into the release folder as well. If some files, such as images, also need to be installed to a web server a *release.sh* script should also be produced.

The Database Administrator (DBA) or release manager can then utilize the *release.xxx* script(s) to run the updates in the QA/Test or Production environment using SQL*Plus, or a similar command line interface. If the release contains DLL and/or DML scripts, downtime is generally required.

Downtime is not mandatory to import Application Express applications, as they can be imported into a Production environment actively being used by end users, as users will retrieve the latest application definition the next time they submit a page. However, it is best practice to disable all user access during the release rollout, to prevent end users getting unexpected errors while only some of the release is implemented.

An easy way to prevent end user access, during rollout, is to update the Application Express application(s) Availability: Status from 'Available' to 'Unavailable' at the beginning of the rollout, and then ensure the application imports are the last scripts run during the rollout. When a new application is imported the Availability: Status will be reset back to 'Available'. In this way, when a user submits a page or tries to access an affected application, during the rollout, they will receive the 'Message for unavailable application' defined within each application.

A new API that will allow the application status to be set programmatically from SQL is planned for Application Express 5.1. For current releases, if you can access the Application Builder in Production, then you can set application availability manually by editing the application properties and setting the Availability: Status.

Implementing Bug Fixes

Wherever possible it is preferable to implement bug fixes as part of a scheduled release. However, for critical, show-stopper bugs, found in existing production releases, it is often necessary to implement a fix as soon as possible. The same basic procedures as outlined in Propagating Releases should be employed for deploying bug fixes into QA/Test and Production environments.

A decision to fix and deliver a bug mid-release should not be made lightly, as it increases risk and can be very expensive to current development efforts. Often, such bug fixes must be implemented twice – once in the production application and again in the current development application.

The Application Express Development Team manages such major bugs by maintaining a separate development environment of the current production release. Bugs are then fixed in this environment. Patch set exceptions are then created, and published to My Oracle Support. If you are utilizing Oracle Database 12c Multitenant Environment, it may be preferable to clone your production PDB to fix the major bugs. Switch Application Express from "Runtime Only" to full development in the cloned PDB, make the bug fix in the cloned environment, and then export the fixed application(s). Alternatively, if your organization has a stage environment, with full development rights, that may be an environment that can be utilized for fixing bugs. Using these techniques, the bug fixes must be reapplied in the current development environment, to ensure a regression is not introduced with the next release.

Unfortunately, most organizations do not have the luxury of a separate environment where bugs can be fixed. Depending on the scope of the fix required, and the amount of affected new development components, there are different techniques which can be employed.

If the bug fix requires changes to only a small number of components within the application, and such components are not impacted by new development, then it may be appropriate to apply the bug fixes directly in the Development environment, within the current application being updated with new functionality for the next release. Once ready, use application component export to only export the changed components. Irrespective of how quickly the bug fix is required in production it is very important to import the components into the QA/Test environment first, and perform regression tests. This is to ensure the bug is fixed, and it has not introduced any regressions, or included any new development not intended for production release.

If the bug fix is more extensive and impacts numerous application components, analysis of upgrades for the new, unreleased version of the application should be undertaken. You need to determine if all new development, not ready for production, can readily be excluded using Build Options. If using Build Options is feasible, then bug fixes can be carried out directly in the current application. Once all new development has an appropriate Build Option and the changes to fix the bug have been implemented, a full application export should be used to migrate the application to QA/Test. However, if not practical to use Build Options, you should check the production application out of your source control, and import it back into the Development environment with a different Application Id. Once the changes have been completed, a full export of this revised application should be performed, and under Export Preferences ensure **Export with Original IDs** is set to **Yes**. This will export the application with the same component IDs as in the original application. It is also critical to make the same changes a second time in the new release of the application, currently being developed for the next release.

Build Options

One of the best techniques to exclude certain unfinished development from an application export is to utilize 'exclude' Build Options. Build Options are used to conditionally include / exclude specific functionality when an application is imported. Build Options can be applied to most application components including pages, regions, items, buttons, processes, validations, dynamic actions, branches, and list entries.

When defining a Build Option, under Shared Components, you can specify that the option exclude the functionality, and can also specify to exclude the component from the export. Therefore, developers can continue to update unfinished components with the appropriate Build Option, as the the application can be exported at any time without including those components. Review Oracle Application Express Builder User's Guide > 24.11 Using Build Options to Control Configuration to learn more about customizing an application export.

Managing Application Express Development

There are many aspects of maintaining large-scale development, with a large development team, using Application Express. Below are a number of different considerations.

Application Pages

When you are building a large solution with a team of developers in Application Express, it is often much easier to manage the deliverables if you break up the overall solution into multiple smaller applications. Defining separate applications for discrete functional areas will reduce the number of developers working on the same application, at the same time. As a rough guide, each application should have no more than 50 pages. This will make it far easier to manage discreet releases and handle bug fixes.[1]

Subscriptions

On the assumption that the solution is broken into numerous smaller applications, you should consider defining a "master" application to hold common components that can be subscribed to from all other applications. Components can include the application theme and theme style, authentication scheme, authorization schemes, plug-ins, and list of values. If all of the other applications use 'Copy and Subscribe' for these components, then they can be maintained in a single application, updated once, and then propagated out to all other applications.[2]

Tracking Progress and Scope

One of the most important aspects of large application development is managing the project timeline, and scope. Team Development within Application Express was designed specifically for this purpose. Project managers and developers can define milestones, features, bugs, and to-dos within team development. To-dos can be associated with either features or bugs and are generally used by developers to break down large deliverables into smaller development activities.

Features, bugs and to-dos can reference an Application Express application and pages. Within Page Designer the number of associated entries will be displayed, making it easy for developers to identify required work for a given page. While there are several project management tools that can be used for this purpose, none has the ability to link directly to Application pages.

---

[1] *Different applications can be configured to utilize the same cookies so that end users can move from one application to another without needing to log in each time. Developers must specify the same Session Cookie Attributes within the current authentication theme in each application*

[2] *Copying and Subscribing is only possible between applications in the same workspace. For this reason, it is preferable to develop all the applications within a large solution in the same workspace, and reference the same "master" application.*

Team Development was introduced to allow the APEX Development Team to manage the deliverables for Application Express releases. The team enters numerous milestones for the different phases of the development, such as drops for Early Adopters, drops for Translation, and so forth. Features and bugs are then assigned to specific milestones, or pushed off to a later release. Features are generally defined in a hierarchy of major functional areas to aid tracking of progress. Team members often enter to-dos, on features they have been assigned, to break the solution into smaller, discrete programming units.

Developer Conflicts

In Application Express 5.0 and above, if two developers are updating the same page at the same time, using Page Designer, then they will receive an error if they update, and save, the exact same attribute as already saved by the other developer.[3] For example, if they were both to update the Title for the same region, then the second developer would receive an error. However, if they were updating two different regions or attributes, then both changes would be saved without error.

To prevent multiple developers updating the same page, developers can easily lock the page(s) they are currently enhancing. When another developer accesses the locked page they will see that the page lock is red. They can still view the page definition but not update any attributes.

When many developers are working on the same application, it is important that developers changing shared components in an application, such as a List of Values, review all the pages where that shared component is utilized, to ensure they don't introduce regressions.

Externalizing Code

Within an application, developers can include SQL and PL/SQL code in numerous attributes and components, such as conditions, validations, and processes. It is best practice to move such code into Oracle Database objects (packages, functions, and procedures) and then call the object from within the application. For example, rather than coding many lines of PL/SQL code directly within a complex validation, the code can be written in a Database function which returns the appropriate error message or a Boolean to the validation.

Using Oracle Database objects, rather than extensive coding within the application, will enhance re-usability across different pages and applications. Maintainability of the business logic is also greatly improved, as there is a single source of code to maintain. External objects should be developed using SQL scripts and checked into source control.

---

[3] *In earlier releases developers are far less likely to encounter such an error, as changes are saved whenever they navigate to the various property pages for each component.*

Developer Skill Sets

Within any moderate to large development team, there is generally a range of expertise and experience across various development languages. The different types of developers can include:

- Basic Developer – Typically have a basic knowledge of SQL, and preferably PL/SQL experience. They primarily use the Application Express wizards, the declarative Page Designer, and various shared components to create and enhance applications.
- Advanced Developer – Generally have been using Application Express for a number of years, and have in-depth knowledge of SQL and PL/SQL, together with JavaScript (JS) experience. As well as mentoring other developers, they are often responsible for prototyping and defining development standards.
- UI Developer – This is a specialist role with expertise in HTML, CSS, and in UI design principles. These experts should be responsible for defining the theme and theme style customizations for all applications. They should be responsible for ensuring a consistent user experience (UX) as well.
- Integration Developer – As applications increase in complexity, such as applications deployed in the cloud that interact with other systems or resources over the Internet, there is a growing requirement for specialists that can provide expertise around REST, SSL, JSON, and other protocols. These experts are normally tasked with defining and implementing data transfer to and from different applications.
- DBA – Oracle Database experts are often critical, especially for large-scale development projects. They should be heavily involved in the design, creation, and management of database objects to ensure in-house standards are followed. For example, ensure that primary key and foreign key constraints are properly implemented, and that appropriate audit columns are included. Another major responsibility for DBAs is to conduct performance reviews and assist developers to deliver performant applications. The large majority of performance issues, related to Application Express applications, is directly related to poorly written SQL implemented by developers within the application. In addition, there may be an occasional need to manage SSL certificates, database wallets, and other aspects of database configuration.

The majority of Application Express developers simply need SQL and PL/SQL experience. Generally, not everyone is an expert in languages such as JavaScript, HTML, and CSS, and not everyone is an expert in User Experience (UX), nor do they need to be experts in these technologies. Therefore, it is best practice to limit the majority of developers to working only with standard Application Express components, and have a small number of experts responsible for implementing JavaScript, HTML, and CSS across all of the applications.

For example, a developer, who is not experienced with JavaScript, may spend an inordinate length of time developing a "cool" JavaScript widget. However, the end result is often not in keeping with the rest of the application, very difficult to maintain, and not a good utilization of development resources. Even worse, the same result may have been very easily and quickly implemented, declaratively, using Dynamic Actions. By having that same developer go to the select experts, then such lost productivity and poor results can be minimized.

Application Exports

In order to prepare a release, applications are generally exported from the development environment. When an application is exported a plain-text, SQL script file is created. A full application export includes the application definition, supporting objects, and shared components (including plug-ins, images, CSS files, JavaScript files and other files which must be managed independently), but does not include any of the underlying database objects or data.

The application export file(s) should be checked into the source control system as part of the deliverables for a new release. This file can then be checked out, and used to import the application into any other Oracle Database, such as QA/Test or Production, which will insert all of the application meta-data into the new Application Express engine schema. When importing the application into QA/Test or Production environments it is strongly recommended that each application is imported with the same Application ID, as used in development.

The export utility can also be used to export any application components individually, such as a complete page, or a region, and so forth, rather than the complete application. However, there is a risk that using component exports may not include all of the required components for a release, such as updated shared components, like List of Values. For application components to be able to be imported into a different environment it is essential that the workspaces in those environments are created from an export of the Development workspace, so the Workspace ID is the same, and that the Application IDs are the same between environments.

Comparing Application Exports

There is an excellent open source product called APEX Diff (https://github.com/OraOpenSource/apex-diff) which creates JSON exports of an application which can be easily compared with other versions of the application. This product is one of the best alternatives for comparing different releases of an application.

Using regular 'diff' tools to analyze the deltas between two application export files, exported from different environments, will raise a very large number of false positives, making it nearly impossible to identify true differences in functionality. The main reason for this is that every component ID is different when an application is exported from a different environment.

Regular 'diff' tools can identify differences between two exports of the same application taken from the same environment at different times, for example two different versions stored in source control. The differences will be represented as changed API parameters, such as in calls to wwv_flow_api.create_page_plug, in the script files. As such reviewing such differences is of limited value, except to very experienced Application Express developers who have an intimate understanding of the underlying meta-data tables, and can correlate differences in API parameters to specific application components. The application statistics at the top of the export files will provide differences in total components, which may provide some value.

Warning: Merging of application export files, or in any way modifying the application export script files is not supported. Updating one of these files, directly with a diff tool or text editor, may cause the import of the application to fail. This may also corrupt your Application Express meta data, which may be unrecoverable.

Planning for Lost Application Development

In any large scale development, it is important to plan for recovering from lost deliverables, such as applications or components that are inadvertently deleted, or a corrupted database. If the Oracle Database, which hosts your Development environment, was corrupted, or a DBA accidently dropped the Application Express Engine schema (APEX_050000 or similar), or a developer wrongfully terminated a critical workspace, or a developer deleted the wrong application, would you be able to restore the environment / applications? How long would it take? How much effort would be required? How much development would be lost?

Archiving Applications

One of the best defenses for lost application development is to backup your applications regularly. Within the Packaged Apps is the APEX Application Archive. This application allows you to archive your applications into APEX$ tables within your development schema, associated with your development workspace. Your archives can be full or incremental backups, and you can specify which applications should be included in either type of backup. You can restore any application from any archive, or download the application through the Application Archive.

The APEX Application Archive automatically tracks application versions and gives you a high level view, showing when each of your apps has been most recently backed up. The APEX Application Archive also lets you specify the number of versions you want to keep of any application and to purge older application versions. By keeping your archives in database tables, the APEX Application Archive insures that your applications will be backed up just like your data. By automating the backup process, the APEX Application Archive makes it easy for you to keep multiple versions of your applications, increasing your flexibility without impacting your productivity.

Command-Line Export Utilities

An alternative to storing application archives directly within your development schema is to back up your applications into your source control system. The Oracle Application Express installation zip file include two Java programs APEXExport.class and APEXExportSplitter.class. Both files can be found under /utilities/oracle/apex. These programs can be used for extracting applications and/or application components directly into source control on a nightly basis. These backups are excellent for disaster recovery. The output from these Java programs are SQL script files that can be used to completely rebuild an entire workspace and/or applications as needed.

The program APEXExport can be used to export Oracle Application Express applications or workspaces from the command-line, without requiring a manual export from the Web interface of Oracle Application Express. The program APEXExportSplitter can be used used to split Oracle Application Express export files into separate SQL scripts. This is useful for management of files corresponding to the discrete elements of an Application Express application. These jobs can easily be set up within a CRON job, that runs a shell script, to automatically export the required applications, and use command line source control tools, such as git, to check the artifacts into your source control system.

You must be in the utilities directory when executing any of the code examples below.

To review all of the available options, run the following:

```
java oracle.apex.APEXExport
```

To generate an SQL script file for each workspace, run the following:

```
java oracle.apex.APEXExport -db localhost:1521:ORCL -user system -password
systems_password -expWorkspace
```

To generate an SQL script file for every application in all workspaces, run the following:

```
java oracle.apex.APEXExport -db localhost:1521:ORCL -user system -password
systems_password -instance
```

To generate an SQL script file for all applications in a single workspace, run the following:

```
java oracle.apex.APEXExport -db localhost:1521:ORCL -user system -password
systems_password –workspaceid 9999
```

To generate an SQL script file for a single application, run the following:

```
java oracle.apex.APEXExport -db localhost:1521:ORCL -user system -password
systems_password –applicationid 31500
```

To generate multiple SQL script files for an application, run the following:

```
java oracle.apex.APEXExportSplitter f31500.sql
```

Example Utilization of Command-Line Export Utility

The Application Express Development Team utilizes Hudson to automate our continuous integration, whereby the complete development environment is exported into subversion and then rebuilt into a new environment nightly. Using built-in Hudson capabilities you can define shell scripts similar to the example below.

The process followed in the script is to first check out the current version of each application from source code control utilizing Hudson's SVN integration features. The application SVN files can be locked or created as required.

```
svn co $SVN_URL $APP_DIR
svn lock –force --message "$LOCK_MSG" "$appFileName" || { touch "$appFileName" && svn
add "$appFileName" ; }
```

After the SVN structure has been established, the APEXExport tool is used to extracted the current version of the application file. Options below remove the date stamp from the export (-skipExportDate) to allow SVN to recognize differences in the exports.

```
java oracle.apex.APEXExport \
      -db $APEXDB \
      -user $APEXDB_UN \
      -password $APEXDB_PW \
      -applicationid $appID\
      -skipExportDate
```

The output file from APEXExport is saved to the current (utilities) directory. Then the file names are adjusted and moved to the appropriate location in the SVN directory.

```
mv -f f${appID}.sql $appFileName
```

Commit the new version of the application to SVN

```
svn commit --non-interactive --message "$COMMIT_MSG"
svn unlock --force
```

Restoring a Deleted Application Component

Should a developer accidently delete an application component, such as a page, then providing the mistake is identified in a timely manner, then a developer should be able to export the application using the **As Of** setting, to create an export from before the component was deleted. That export file can then be imported into Development, restoring the lost component(s). However, this will also overwrite any other development performed during the interim. This utility uses the Oracle Database *dbms_flashback* package, and utilizes the Oracle Database undo logs. The length of time a developer can go back is determined by the Oracle Database startup parameter *undo_retiention*, and how quickly the undo tablespace is archived.

Restoring a Corrupted Development Environment

On the assumption that your DBAs run nightly backups of the complete Oracle Database, then with effort, the Development environment can be recovered in all of the above scenarios. Performing regular backups of the workspaces and their applications into source control will make recovering from such disasters far simpler and quicker, minimizing lost developer productivity.

If your database is corrupted, and then the whole database is restored, then your Application Express environment should be fully functional once the database is restored. However, if the APEX_050000 (or similar) schema needs to be recovered from the full database backup, then it is far more complex without Application Express exports as a backup. To restore the schema, you would need to download the appropriate version of Application Express from the Oracle Technology Network, and then install Application Express. If proper backups are available, you could simply use the workspace and application import scripts to recover the environment. If not, generally, the DBA will need to stand up a new instance of the Oracle Database and recover the full backup into that new environment. All of the workspaces and applications would need to be exported from the newly restored database, and then imported into the recovered Development environment.

For lost workspaces and applications, if appropriate backups are not available, the DBA would again need to stand up a new instance of the Oracle Database and recover the full backup into that new environment. The missing workspaces and/or applications could then be exported from the newly restored database, and imported into the Development environment. Recovering the schema directly into the Development environment, the Application Express engine will not be correctly configured, and it will overwrite all existing applications in all workspaces. This will lead to the loss of any development performed since the backup was taken.

Summary

Oracle Application Express is a highly productive way to build Web based applications whether using one developer, or a team of developers. Given the differences between how developers work with applications in Application Express, as opposed to file-based languages, it is important to understand and implement the best practices for developing and deploying applications, across multiple environments for numerous releases.

These recommendations will have value in any Application Express project, but they are especially valuable for larger teams developing more complex and more critical applications, not least because they enable Application Express projects to be managed in the same way as those based on other technologies.

Consideration should be given to the following:

- All database objects should be maintained using SQL script files;
- Use SQL script files to implement all required database definition language (DDL) and data manipulation language (DML) changes;
- Install runtime-only Application Express into QA/Test and Production environments;
- Export Development workspaces and import them into QA/Test and Production environments;
- Have developers ensure all release deliverables are checked into source control, and a single *release.sql* script file is produced for the DBA to utilize to build the release into QA/Test and Production environments;
- Wherever possible implement bug fixes as part of a scheduled release;
- Utilize Build Options to conditionally exclude unfinished development from an export;
- Separate functionality into smaller discrete applications rather than a single large application;
- Define a "master" application and have all other applications subscribe to the common components;
- Externalize database logic into Oracle Database objects, rather than having extensive code within application attributes and components;
- Export complete applications, rather than application component exports, except potentially for high priority bug fixes; and
- Use APEX Application Archive and/or APEXExport.class and APEXExportSplitter.class for regular application back ups.

ORACLE

Oracle Application Express Version Control

May 2016

Authors: David Peake
Contributors: Joel Kallman, Carsten Czarski, Jules Lane, Martin D'Souza, Scott Spendolini
Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together