Automatic promotion and versioning with Oracle Data Integrator 12c

Jérôme FRANÇOISSE
Rittman Mead
United Kingdom

Keywords:

Oracle Data Integrator, ODI, Lifecycle, export, import, smart export, smart import, repositories, scenario, load plan, package, versioning, promotion, ODI Tools, ODI SDK, source control, SubVersion, Git.

Introduction

Oracle Data Integrator 12c is a great enterprise tool providing the ability to work efficiently on different environments. It might be the traditional architecture with Development, Test and Production environments but there are also a lot of other combinations. Oracle Data Integrator stores the metadata in repositories. When promoting code from one environment to another, this metadata needs to be copied into other repositories. At the same time, it is always a good practice to keep track of every changes made by the developers to be able to restore an earlier version of the code. This can be done using the built-in versioning functionality of Oracle Data Integrator or using a third-party tool. This paper aims at describing the Development Lifecycle. We will review some common architectures and the different ways to promote and version the code.

Repositories

Oracle Data Integrator stores all it's metadata in repositories. The topology – containing the connections to the datasources and the definition of agents and repositories –, the security and the versioned objects are stored into a master repository. A master repository can have one or more work repository attached to it, where all the other metadata resides.

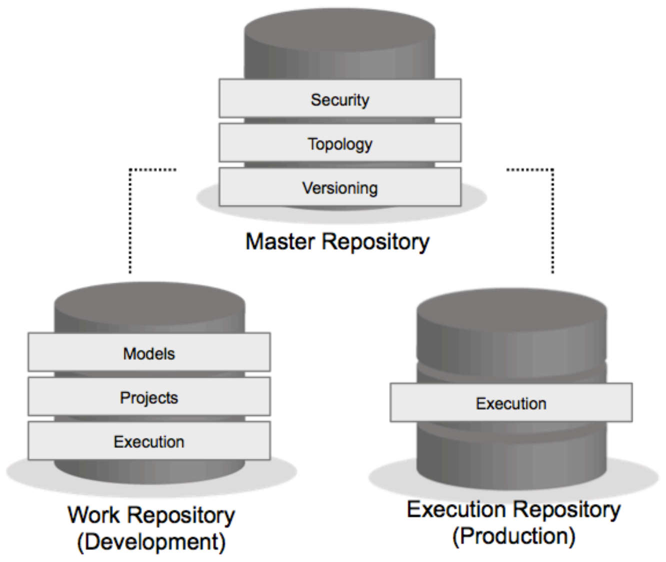


Illustration. 1: Oracle Data Integrator repositories.

The work repository comes in two modes offered at creation time: Development repository or Execution repository. The former can store all the metadata about projects – including mappings, packages, procedures, variables, sequences, knowledge modules, … –, models – holding the data structures – and the operator – holding the logs of previous execution and the list of available scenarios and load plans. An execution repository can only hold metadata about the operator. It is therefore meant only to execute Scenarios and Load Plans and view the result of the execution.

Development Lifecycle

The typical way of developing with Oracle Data Integrator is to have developers working only on a Development repository, using it as the Development environment. In this place, they can develop their code and test it again data populated especially for that purpose. Once they have a consistent set of objects ready to be promoted, they can generate scenarios of their objects and optionally sequence the execution into a Load Plan.

These objects can be promoted to an Execution repository. Usually the Test environment or the Production environment are execution-only, to avoid developers to fix bugs in the wrong environment which would lead in out-of-sync repositories. Once the Scenarios and Load Plans have been promoted, they can be executed or scheduled and the result of their execution is stored into the same repository. Depending of the architecture and the procedures of each company, there might have more than one promotion required to reach the production environment. Code reviews or Project Owner approval can also be pre-requisites for such a promotion.

If a bug or a change request occurs, the developers have to modify or edit the objects back into the development repository and start a new iteration of the lifecycle.

Thanks to this approach, the code currently sitting in Production can run successfully while new development iterations can take place on the same objects in the Development environment. However, once changed in the Development environment the underlying objects – mappings, packages, … – are not reflecting the scenarios running in Production. As a scenario is a complete black-box version of an object, it is not editable. Therefore is important to always keep track of the underlying object code associated to every scenario version. This is where the concepts of versioning and source control can answer our needs.

Using either the out-of-the-box versioning capabilities or a third-party tool, developers can create a version of their objects every time they want to have snapshot and at least before each release. They need to keep track of which version of the object is related to which scenario version in order to be able to rollback to a certain point in time.

Architecture

The architecture can be simple, with two environments sharing the same master repository as illustrated in *Illustration 1*. While this can be efficient for a small team working with flexible infrastructure, this might not work at all in a more complex company.

Typically, data integration jobs need to be validated by a testing/UAT team before making their way to Production. This requires adding a new Testing environment.

Another frequent requirement is to totally isolate Production from the other environments. This might be done through a firewall and this implies to have a separate master repository for the Production environment. In this case, the Logical Topology needs to be replicated to match perfectly in all environments. With this new separation, testing jobs on the Test Environment is not enough to prove that it will work on the Production environment, because an execution on a different master repository has never been done. Adding a new PreProduction environment with it's own master repository is the answer to this last issue.

Having all these environments is great way to have reliable integration jobs, but it's at the expense of the flexibility. If a bug is discovered in Production, going through all the steps – Development, Test, PreProduction and finally Production – takes a long time. To have a faster response, a Hot Fix environment can be added and linked directly on the PreProduction master repository.

Finally, in order to support continuous integration and smoke testing, a last environment can be created and attached to the Development/Test master repository. Each night, all the scenarios marked as ready can be promoted from Development to this environment and some smoke testing execution can be scheduled. Thanks to this, a broken job can easily be identified and quickly reported.

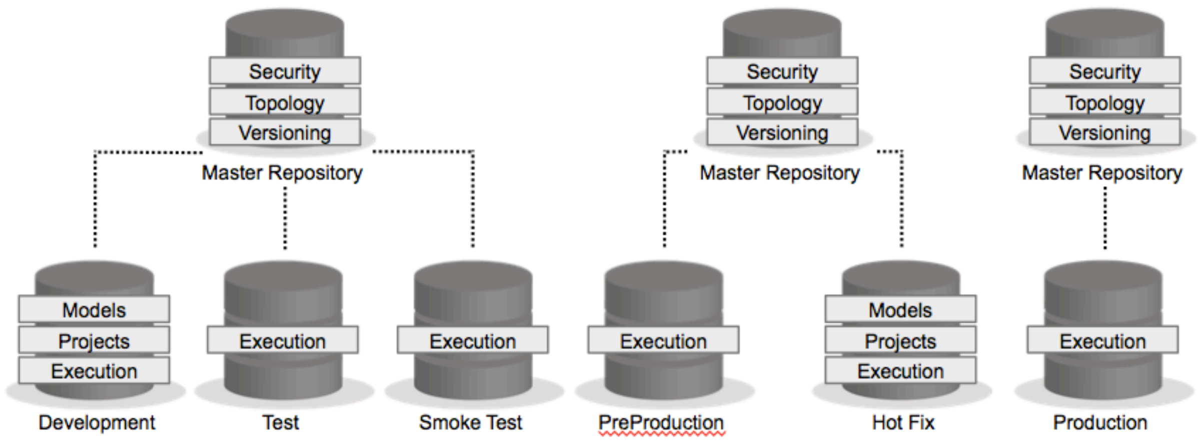


Illustration. 2: Enterprise-level Architecture.

Every company is different in term of flexibility, process and infrastructure, so it's important to find the architecture that corresponds to its needs, from the simple one with two environments to this complex one with six environments and three master repositories.

Promotion

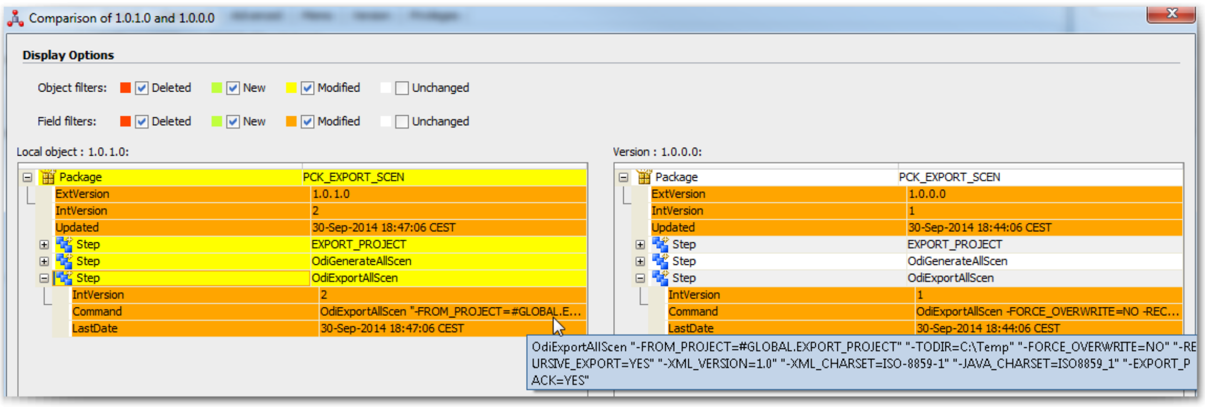
Promotion from one environment to another is usually done by exporting objects from a repository and importing them in another one. It can be objects, such as Mappings, Packages or Procedures, to promote to another development environment. Or it can be Scenarios – a fixed snapshot of the code ready for execution – and Load Plans to promote to an execution environment.

In the first case, Oracle Data Integrator provides a really useful functionality called smart export and import since the 11.1.1.6 release. The smart export allows picking some objects as part of the build package and Oracle Data Integrator Studio will find all the dependencies to other objects and add it to the export. When importing, a dialog box appears to find a resolution for conflicts with existing objects. The user can chose to keep the existing one, import the new one or merge the two together.

When exporting Scenarios and Load Plans, the typical export and import are used. It is recommended to use Load Plan and Scenario Folders to organise the scenario per project and possibly per release. The agents can be updated to take the newly imported scenario's schedules into account or new schedules can be added.

Versioning

The 11g release introduced a built-in versioning mechanism within Oracle Data Integrator. New version of Interfaces or Mappings, Packages, Procedures, User Functions, Variables, Sequences and Scenarios can be stored into the master repository with a comment to explain the changes made. As this is stored into the master repository, other environments attached to it can also access it. The user also has the ability to create a Solution and drag a whole project into it. This will results in the creation of a new version for every object in that project. Versions of objects can then easily be compared, restored or deleted.



Comparison of 1.0.1.0 and 1.0.0.0

Display Options

Object filters: 🟥 [✓ Deleted] 🟩 [✓ New] 🟨 [✓ Modified] ☐ Unchanged

Field filters: 🟥 [✓ Deleted] 🟩 [✓ New] 🟧 [✓ Modified] ☐ Unchanged

Local object : 1.0.1.0:

Package	PCK_EXPORT_SCEN
ExtVersion	1.0.1.0
IntVersion	2
Updated	30-Sep-2014 18:47:06 CEST
Step	EXPORT_PROJECT
Step	OdiGenerateAllScen
Step	OdiExportAllScen
IntVersion	2
Command	OdiExportAllScen "-FROM_PROJECT=#GLOBAL.E...
LastDate	30-Sep-2014 18:47:06 CEST

Version : 1.0.0.0:

Package	PCK_EXPORT_SCEN
ExtVersion	1.0.0.0
IntVersion	1
Updated	30-Sep-2014 18:44:06 CEST
Step	EXPORT_PROJECT
Step	OdiGenerateAllScen
Step	OdiExportAllScen
IntVersion	1
Command	OdiExportAllScen -FORCE_OVERWRITE=NO -REC...
LastDate	30-Sep-2014 18:44:06 CEST

```
OdiExportAllScen "-FROM_PROJECT=#GLOBAL.EXPORT_PROJECT" "-TODIR=C:\Temp" "-FORCE_OVERWRITE=NO" "-RE
URSIVE_EXPORT=YES" "-XML_VERSION=1.0" "-XML_CHARSET=ISO-8859-1" "-JAVA_CHARSET=ISO8859_1" "-EXPORT_P
ACK=YES"
```

Illustration. 3: Comparison between two versions of a package.

An alternative is to use third-party source control tool. The two tools the more widely used at the moment are Subversion and Git. They are both dealing with files directly and they allow to add files to the source control mechanism. The developers can commit they changes as much as needed and it will results to a new version in a central repository. They can both also handle conflicts if more than one developer edited the same code. Git goes even further allows the developers to see each version of their changes in a local repository so they can create version more often even if they are not done with their objects. With Git, a branch can also be created for each team or even each new feature. After the work is done in a branch, it can be merged back into the master branch. This feature becomes handy if several development teams work on the same objects in different Development repository. When they will want to promote the code in a centralised Development repository, they will need to resolve the conflicts when merging their branch.

The need to automate

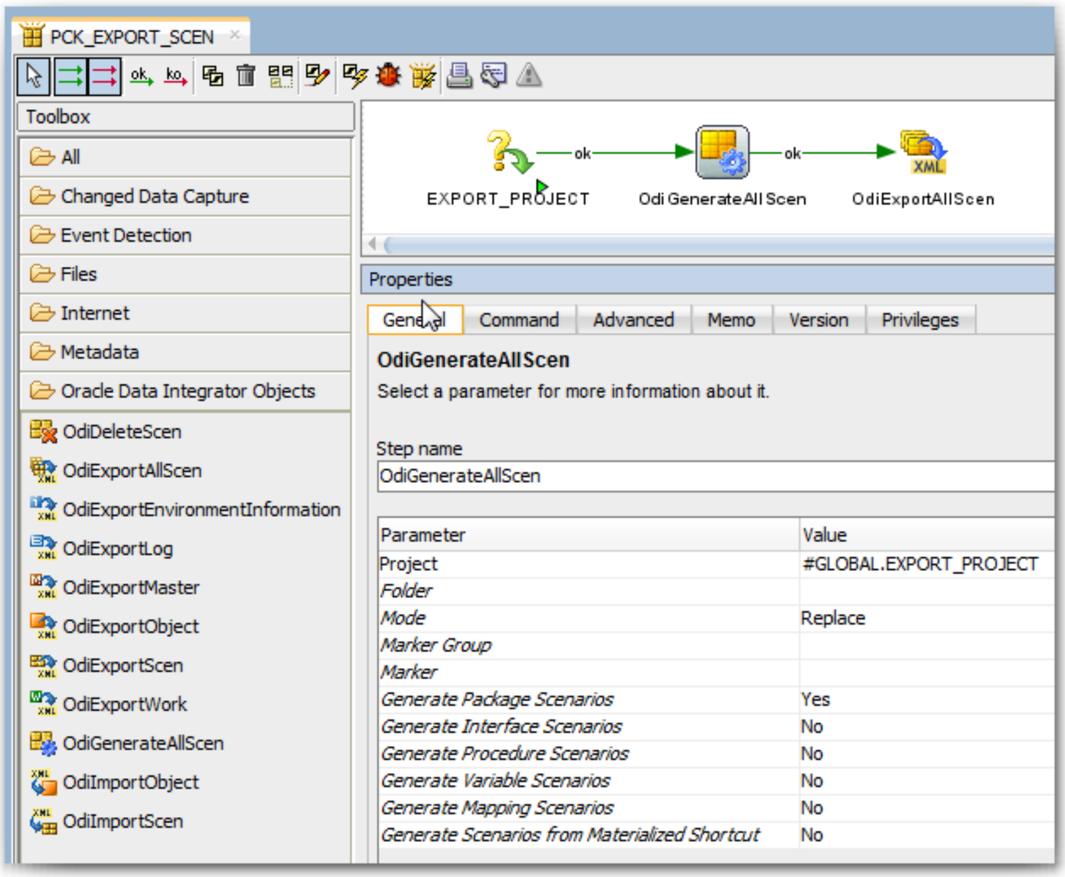
Promoting from one environment to another implies to always repeat the same tasks. In the IT field, this is typically something we will try to automate. After a fix cost for automating the task, some time is spared every time objects needs to be promoted. Instead of generating a scenario for all the objects that need to be promoted, exporting the scenarios, disconnecting from the repository, connecting to another repository and then importing the scenarios, it's easier to run a script.

Repeating tasks is also error-prone. It's easy to reconnect to the wrong repository and execute in Production during busy what was supposed to be tested and validated on the Test environment. It's also a common error to select the wrong mode when importing and duplicate scenarios with a new internal idea instead of insert or update it. Automation removes these risks.

Automatic Promotion

The most common way to automate the promotion is to use the ODI Tools available out-of-the-box with Oracle Data Integrator. These ODI tools can be used in a package, in a procedure or even called from the command line interface.

Three ODI tools will catch our attention. The first one, OdiGenerateAllScen, allows to automatically generate new scenarios for all the objects belonging to a project or to regenerate existing scenarios. Additional parameters can be set to specify a single folder or only the objects having a certain marker set. It is also possible to generate scenarios for only one type of objects : packages, mappings, procedures or variables. The second ODI tool is OdiExportAllScen and similarly to the previous one, the user can select objects belonging to only one project, folder or marker. For each of these parameters, it is possible to pass the value of a variable. This way it is possible to reuse the same code to generate and export the scenarios for different projects.



OdiGenerateAllScen

Select a parameter for more information about it.

Step name: OdiGenerateAllScen

Parameter	Value
Project	#GLOBAL.EXPORT_PROJECT
Folder	
Mode	Replace
Marker Group	
Marker	
Generate Package Scenarios	Yes
Generate Interface Scenarios	No
Generate Procedure Scenarios	No
Generate Variable Scenarios	No
Generate Mapping Scenarios	No
Generate Scenarios from Materialized Shortcut	No

Illustration. 4: Package generating and exporting all the scenarios for packages within a given project.

The last ODI tool needed for the promotion is OdiImportScen. This should be executed on the target repository and it will import the xml files generated by the two previous steps.

Another way to promote code is to use the ODI SDK. Thanks to the public API, almost everything done in ODI Studio can also be scripted. Similarly to what we can do with the ODI Tools, the goal here is to generate the scenarios, export it and then import it in another repository.

This can be done using the following API calls :

```
/*
(...) Connection to the source repository
(...) Creation of a collection of mappings
(...) Instanciation of an OdiScenarioGeneratorImpl object
*/

// Generating Scenarios
For(Object mapping : mappings) {
    Mapping odiMap = (Mapping) mapping;
    String scenName = odiMap.getName();
    OdiScenario newScen = gene.generateScenario(odiMap, scenName,
newVersion);
}
```

```
/*
(...) Creation of a collection of the generated scenarios
*/

//  Exporting Scenarios
for (Object scen : scenarioCollection) {
   OdiScenario odiscen =(OdiScenario)scen ;
   export.exportToXml(odiscen, ExportPath, OverWrite,
RecursiveExport, Encoding);
}


/*
(...) Connection to the target repository
(...) Creation of a list of XML files.
*/

// Importing Scenarios
for (String filename : XMLFiles) {
   import.importObjectFromXml(
import.IMPORT_MODE_SYNONYM_INSERT_UPDATE, filename, true);
}
```

This code can be compiled into java classes but it can also be directly use in a Groovy script. Since the 11.1.1.6 release, Oracle Data Integrator can directly run Groovy script from ODI Studio. Groovy script can also be integrated in a procedure. This increases the flexibility of development and there is no need to run an external script.

Automatic Versioning

Having an easy to promote the code is useful, but it would be even better to keep track of the code every time it is promoted. This way we can always restore an earlier version of the code, before the promotion. By automating it, we don’t rely on the developer to do it and we are sure nothing is missing.

Unfortunately, there is no ODI Tool related to object versioning. There is not any ODI SDK calls related to the internal versioning either. It means that the only way to automate the versioning is to use third party tools.

Using Git, it’s easy to add all the files from a directory into the local Git repository, then commit it with a message and finally, push it to the centralised Git repository:

```
git add . -A
git commit -m "My commit Message"
git push
```

This code can be added to the automatic promotion mechanism built earlier. For the ODI Tools method, a command-line call with the above code can be added as another package step or procedure step. For the ODI SDK method, this git code can be easily executed on the OS using Groovy or JGit and JavaGit can be used to use a direct Java API.

One step further

Going one step further, we could replicate the same approach as Git does by having a local and centralised repository, but for Oracle Data Integrator this time. By giving a separate development repository for each development team, the developers can have more privileges on it and work without having any object locked by other teams. They also have their own operator and can see their execution in an easier way. Once they are ready to promote their code, it first goes into the centralised Development repository where conflicts are resolved before being promoted to the Test environments.

Unfortunately, building new repositories and replicating the existing metadata from the centralised repository for every new project is a tedious task so it might not be worthy to go that way. Unless…

Coming Soon…

Unless it is included as a new feature in Oracle Data Integrator. During the Oracle Open World 2014, the Oracle Data Integrator roadmap mentioned a new Lifecycle Management feature for the 12.2.1 release. This will allow the easy creation of new repositories for each development team and a mechanism to resolve conflicts when merging in the centralised repository. It will initially support Subversion, maybe other third-party tool will be supported later.

Customers already running Oracle Data Integrator 12c can wait a few more months to have this automatic promoting and versioning. And 11g users now know how to use ODI Tools and ODI SDK to automate it.

Contact address:

Jérôme FRANÇOISSE
Rittman Mead
Moore House, 11 - 13 Black Lion Street
BN1 1ND, Brighton
United Kingdom

Phone:	+44(0)12-73911268
Email:	jerome.francoisse@rittmanmead.com
Twitter:	JeromeFr
Internet:	http://www.rittmanmead.com/