Volume

1

Exodus

Map⏵Document⏵Migrate⏵JSON & Tables

User Guide

Exodus

User Guide

© IHS Markit

Developed By Christian Leigh

Table of Contents

[What can Exodus do? 1](#_Toc10818671)

[Requirements 2](#_Toc10818672)

[Terminology 3](#_Toc10818673)

[Application Anatomy 6](#_Toc10818674)

[Main Window 6](#_Toc10818675)

[Define Migration Window 7](#_Toc10818676)

[Manage JSON 8](#_Toc10818677)

[Getting Started 9](#_Toc10818678)

[Using the Tools 12](#_Toc10818679)

[Running a migration 12](#_Toc10818680)

[Running a migration - Parameters 13](#_Toc10818681)

[How to kill a migration. 14](#_Toc10818682)

[Performance Evaluation 14](#_Toc10818683)

[Relating JSON lines. 15](#_Toc10818684)

[Configuring Exodus 16](#_Toc10818685)

[Manual Configuration 16](#_Toc10818686)

[on\_etl\_params 17](#_Toc10818687)

[pre\_etl\_substitution\_values 17](#_Toc10818688)

[SQL\*Loader 18](#_Toc10818689)

[EXAMPLE Loader control file 19](#_Toc10818690)

[Installation Instructions 20](#_Toc10818691)

[Pre Installation Steps 20](#_Toc10818692)

[Production Systems 23](#_Toc10818693)

[Installing EXODUS schemas 24](#_Toc10818694)

Chapter

1

What can Exodus do?

Learn what Exodus does and familiarize yourself with the basic terminology.

E

xodus is a mapping, documenting, and migration tool especially designed with Oracle PL/SQL developers in mind. If you are comfortable with Oracle PL/SQL then all your mapping and migration code can be constructed here without the need for non-Oracle external tools. There is no magic “black box” behavior. All of the auto-generated code (also in PL/SQL) is made available and viewable from within the tool, and because it’s in a language that seasoned Oracle developers are expert in, there will be no unexpected surprises with the technical approach.

It’s entirely possible to use Exodus to “*just”* document your mapping efforts (it will certainly be far less stressful and error prone than using a spreadsheet to capture mappings and relationships). With just a basic understanding of the tool tables you can write your own SQL reports to get information about how much you’ve mapped (in other words, report on SOURCE data that you have understood enough to express a connection to the target end state) in your migration.

Exodus can be used to do far more than documenting the migration. Exodus can form the core nucleus of your migration where all of the steps required to take data from a set of source tables to a set of target tables in multiple schemas can be defined and coded. Exodus can map table-to-table and table-to-JSON documents. You will be able to capture comments, questions, value translations, functions, and even SQL code to drive your migration.

There is an extensive run framework (including logging, introspection, and debugging instrumentation) to execute your migration to any level of concurrency that your hardware and licenses can support to maximize your compute resource.

# Requirements

🖳

Exodus is designed for Oracle systems. The mapping tool is developed using Oracle Forms 11g and above. You will need an Oracle Database (to contain your source data and to receive the target data (migrated data) and to store the metadata that the tool produces). You will also need one forms server running either locally on your PC or from a centralized server. The database that executes the migration can be hosted in the cloud or on premises. The cloud is an excellent way to boost migration performance because you can elastically “scale up” your compute requirement for the migration “heavy lifting”, and then “scale down” to the modest compute requirements required for day-to-day running.

Oracle corporation allow downloads of all the database tools, and middleware necessary to evaluate the software. Of course if you use this application for production systems you will need the appropriate licenses.

# Terminology

###### See Glossary…

###### The glossary towards the end of this manual will give a useful guide.

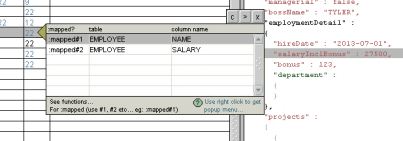
As with any system or methodology EXODUS has its own terminology. Although concepts like **SOURCE** and **TARGET** are universal, there are others that might not be. This section will discuss the key concepts to get you up to speed quickly.

Every migration has a **SOURCE** and a corresponding **TARGET**. In EXODUS the **SOURCE** is “staged data” that represents the starting data before any processing has been done. If your staged data comes from an external non-Oracle system, then the recommended way to get that data onto your system in the most cost effective (inexpensive) way is to use Oracle’s SQL\*Loader tool. There are two types of **TARGET.**  The storage **TARGET** is typically a table (after all, Oracle stores data in tables). **TARGETs** can also be JSON documents. A JSON document will ultimately become a value for a table column (sometimes described as a PAYLOAD) that holds the JSON document; often as a CLOB or VARCHAR2(32767). In EXODUS the **TARGETs** are always expressed as JSON (even tables – Tables appear as a JSON representation of a target table).

**SOURCE** is shown on the left hand side of the EXODUS tool window, and the **TARGET** on the right hand side of the main tool window {See: [Application Anatomy : Main Window](#_Main_Window)}.

**Map/Rules Group**: With any target table, or JSON document there might be more than one use case for mapping to the same target. A good example of this would be a target JSON document that has exactly the same shape, but each use case sources its values from 2 completely different tables (for completely separate migration steps), for example Historical Data and Current Data. Current data could be derived from source table A, but historical data could derive from source table B. They might end up in the same target table (or different target tables), the JSON payloads would be constructed differently because they are mapped differently (In other words the data is sourced differently). The migration definition specifies what **Map/Rules Group** to use (see EXODUS – Define Migration).

**Map & Un-Map**: “To **Map**” means to create a connection between the source table column and a target JSON line. “To **Un-Map**” means to break that connection. A target JSON line can be mapped to multiple source columns (even from multiple source columns from multiple tables – when more than one source table/column is used then it is known as a multi-mapping).

**Multi-Mapping** means to create a connection to more than one source column. **Multi-Mapped** targets can ref. columns in comment code by using # (i.e**.:mapped#1** , **:mapped#2** etc.)

**Stored Document / Stored Fragment / Table Document**: These are all **TARGET** documents. A **Stored Document** is a completely formed JSON document that will be used as a template (or “document by example”) for a mapping target. A **Stored Fragment** is also a mapping target, and is a piece of a JSON document that can be included into a **Stored Document** based upon some Boolean rule (note this “Boolean” is expressed as a ‘TRUE’ ‘FALSE’ VARCHAR2). A **Table Document** is the target table merely expressed as a JSON document (you can only map to column names in this document).

**Related Document**: A related document/line is a way of sharing any mapping made across related JSON documents. Once related, any change to a mapping is made simultaneously to the related mappings. For example you might have a document that has some boilerplate attribute values that are arrived at in the same way from the same source tables and rules, or maybe they’re static values supplied by a fixed value. By relating these target attributes you can alter them all with one edit instead of repeating the change in all the targets.

**Function Comment**: There are 7 types of comment that can be set against a mapping. Three of these comment types can contain code. Array ([ ]), Lookup (LK), and Function (FN). You are not compelled to put code in the comments; you can put anything you like. However if the function begins with an equals sign then it is considered code, a lookup must start with a SELECT or a WITH clause, if it doesn’t then it will just be a comment, and it won’t be considered code.

**Manage JSON**: Used to Add, Update, or Delete Target JSON “documents by example”. If you want to change a JSON document that already has mappings then the tool allows you to re-map either semi manually or automatically so that you don’t lose any of your work made in those mappings. The automatic remapping cannot easily distinguish between attributes that have repeating identical attribute names, so semi-manual re-mapping may be required in some instances, although even that is preferable to mapping from scratch. Table JSONs can also be automatically added.

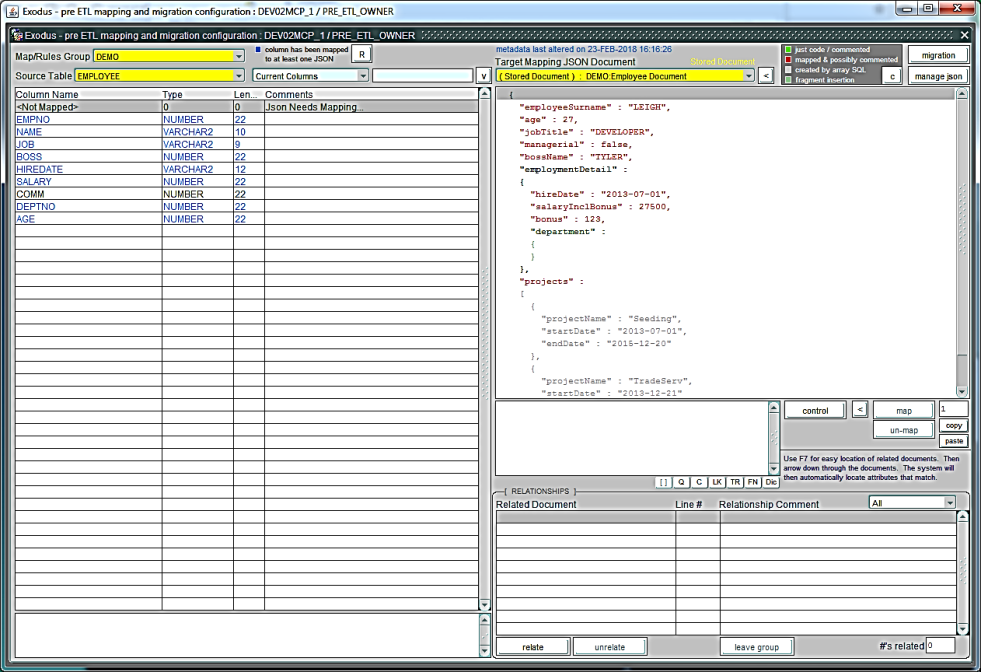
**Migration Grouping**: A way of grouping steps together for specific migrations. The launching of a migration requires that you specify which migration you mean to launch (in other words which Migration Group), and the processing of the migration will normally execute sequentially starting at step 1 of the **Migration Group** and continue to the specified end step. You can also configure **Migration Groups** to execute if certain error conditions arise (for example a **Migration Group** could be a set of steps necessary to return the target schema to a known state on the event of an error).

**Contexts**: There is ONLY one PRE\_ETL\_OWNER schema on an instance; this is where the metadata for all migrations are maintained, and defined. Your design may require more than one target schema for a given migration (A testing schema target, or a production target for example). One way to achieve this would be to have separate migration groups (and mapping rule/groups) for each use case. That would be unwieldy, error prone, and it would necessitate much copy pasting of behavior). Rather than “hard code” these **TARGET** values into your migration you can use ${} variables for substitutes. By doing this you can dynamically change these values for any migration at the point of execution by specifying the context in which you want the migration to run. One way to think of **Contexts** is as a grouping of substitution values.

# Application Anatomy

BELOW are the three main areas of the tool that allow the migration designers to configure the metadata.

## Main Window



**Map Rules Group**

**Search Option**

* All Tables
* Current Columns
* All Tables & Columns
* Targets/Comments/Migration

**Search Value**

Wild Card can be %

EG: TEST%A

**Target JSON**

Target can be JSON Doc. / Fragment or Table

**Colour Key**

JSON documents…

Mapped to Source: Red

**Define Migration Steps**

**Manage JSON**

Add, Modify, Delete

JSON & Remap

**Source Columns**

Double Click to view mapping usage. Blue lines are mapped lines.

**Source Table**

.

**JSON Document**

Attributes can be mapped, commented, or a combination of both.

**Relationship Management**

Relate Fragments

Relate Attributes

**Comment/Code**

Capture comments and *CODE*.

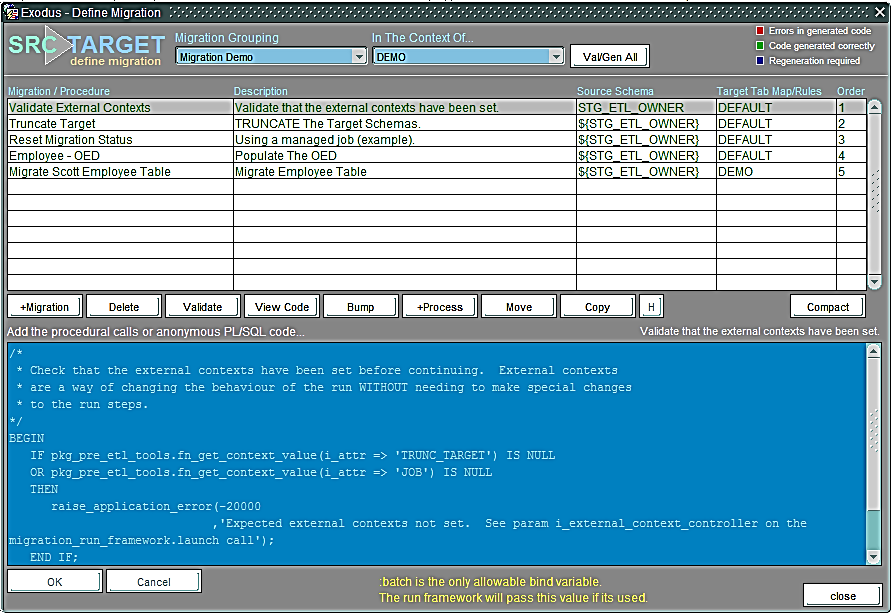
**Leave Relationship Group**

.

**Map / Un-Map**

## Define Migration Window

This is shown when the migration designer presses the migration button from the main window.



**Migration Steps**

All the steps for the chosen Migration Group.

**Migration Group**

Group your logically related migration steps together.

**Context**

The Context can affect the compilation status.

**Validate/Regen**

Compiles and validates the Migration Group.

**Status Key**

Red: Invalid

Green: Valid

Blue: Unknown

**Compact**

Compact the step order numbers to monotonically increasing values.

**Hint**

For auto generated migration steps. Useful for specifying optimiser hints. (Red if used)

**Copy**

Copy a migration step to within the same Migration Group, or to another Migration Group.

**Move**

Move a migration step to another group.

**Add a Process**

Add a manual PL/SQL step.

**Bump**

Increase the step number for the highlighted step and all those below it. Useful for making space to insert a new step.

**View Code**

View the code in a larger window. Useful to examine auto-generated code. The code view window allows convenient copying of the code.

**Validate**

Validates the highlighted step.

**\Delete**

Deletes the highlighted step.

**Add Migration Step**

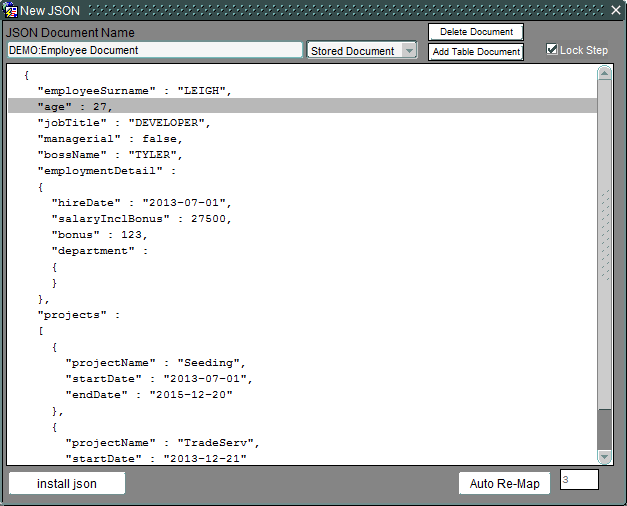
Adds a migration step.



## Manage JSON

This is shown when the migration designer presses the manage JON button.





**Install JSON**

Installs the JSON. Note that if you want to retain any mapping for modified JSON, then you will need to Re-Map BEFORE installing.

**Auto Re-Map**

Instead of manually arrowing down through the JSON, and pressing RETURN/ENTER to copy over the mappings from the target. Auto-Remap performs this task for you.

Use in conjunction with the Lock-Step.

**Add Table Document**

Add tables from schemas defined in the parameters table;

SELECT \*

FROM pre\_etl\_params a

WHERE a.param\_name = 'ACCESSIBLE\_SCHEMAS'

**Lock Step**

This needs to be set for Auto-Remap. When in Lock Step mode, navigation in this window is reflected in the target document too.

**Document Name**

The name of the document being added or amended.

**Type of document**

* Stored Document
* Table Document
* Stored Fragment

**Delete**

Delete the document.

**The JSON document**

Pressing RETURN or ENTER will cause the highlighted line to copy over the mappings and comments of the highlighted line in the target document.

Typically used with Lock-Step mode.

Chapter

2

# Getting Started

1. These getting started steps assume you have installed the EXODUS code for both the database and Oracle Forms tools. See installation instructions.
2. Load your source data into an Oracle database. See: [SQL\*Loader](#_SQL*Loader)

You need to get the SOURCE tables created in the STAGING schema.

You can name the STAGING schema whatever you like, although you may wish to use a naming convention. For example: PROD\_STAGING. This is so that you can also have a DEV\_STAGING and a QA\_STAGING for example. You can use Contextual Substitution Values in your migration code instead of real schema names. Using Substitution Values is a good way to avoid hard coding values that might need to change depending on the context under which you wish to run a migration.

1. Programmatically populate the EXODUS tool tables that store the Source Tables and Columns. You can do this either by writing directly to the tables (there are only 2 to consider; **pre\_etl\_db2\_columns** & **pre\_etl\_db2\_tables**), or you can use the utilities in PRE\_ETL\_OWNER). One advantage to populating the tables directly is that you can add table and column comments. The supplied procedures (shown below) don’t do that. Alternatively you can use the supplied procedures and then back-populate any comments using a script.

BEGIN

pkg\_pre\_etl\_tools.pr\_add\_schema\_to\_mapping\_tool

(i\_schema\_name => '<YOUR SOURCE SCHEMA>'

,i\_tableset\_name => '<A TAG TO NAME THE SET>'

,i\_delete\_existing\_in\_set\_bool => TRUE);

END;

/

Or

BEGIN

pkg\_pre\_etl\_tools.pr\_add\_table\_to\_mapping\_tool

(i\_owner\_name => '<THE SOURCE SCHEMA>'

,i\_table\_name => '<A TABLE NAME>'

,i\_tableset\_name => '<A TAG TO NAME THE SET>'

,i\_delete\_existing\_in\_set\_bool => TRUE);

END;

/

1. Start the EXODUS Mapping Tool.

An example URL might look like this:

<http://mypc:7001/forms/frmservlet?form=C:\forms\PRE_ETL_MAPPING.fmx&userid=pre_etl_owner/pre_etl_owner@migration_db&quiet=YES&separateFrame=True&background=NO&width=1265&height=1000&allowAlertClipboard=false&Logo=%22no%22>

Oracle Forms starts up a Java Applet that runs the tool.

1. If this is the first time you have used the tool then there will be no JSON targets. You will need to add a JSON target. Click the manage JSON button. 

If you are adding a JSON payload target (In other words a DOCUMENT that you want to migrate the SOURCE data to, then you need to paste (or type) in the document. These are often referred to as a “document by example”. For example from the DEMO:

{

"employeeSurname" : "LEIGH",

"age" : 27,

"jobTitle" : "DEVELOPER",

"managerial" : false,

"bossName" : "TYLER",

"employmentDetail" :

{

"hireDate" : "2013-07-01",

"salaryInclBonus" : 27500,

"bonus" : 123,

"department" :

{

}

},

"projects" :

[

{

"projectName" : "Seeding",

"startDate" : "2013-07-01",

"endDate" : "2015-12-20"

},

{

"projectName" : "TradeServ",

"startDate" : "2013-12-21"

}

]

}

You will need to give a name for the document. It’s suggested (but not necessary), that you define a naming standard for your documents. A good way would be to prefix the project name. For example “DEMO: Employee Document”. Set the document type as a **Stored Document**. Then press the load JSON button.



If you want to add a TARGET table (the table will be rendered as JSON), then you should use the Add Table Document button. You can define what schemas make their table definitions available in the PRE\_ETL\_PARAMS table (ACCESSIBLE\_SCHEMAS).

1. You will then be ready to either accept the document by pressing the **install JSON** button, or you can (if you have another document already loaded) re-map or copy the mappings from another document.
2. Once you have loaded your document you can map to it. Select a line in your JSON document, and then choose a SOURCE table/column. Press the map button. If you map to more than one source you will be given an option to start a multi-mapping, or overwrite the existing mapping (if any).
3. It is strongly recommend that you install the DEMO migration. The DEMO migration takes the classic SCOTT/TIGER database, and migrates it to a Target database in which the contents of the source database are represented as JSON documents.

Chapter

3

# Using the Tools

## Running a migration

ONLY ONE MIGRATION CAN RUN AT A TIME.  The system will prevent you from running more than one simultaneously.  This is to do with how the migration framework shares state across multiple threads.

BEGIN

   migration\_run\_framework.launch(i\_batch                        => 123

                                 ,i\_migration\_group              => 'MY-MIG'

                                 ,i\_context                      => 'DEV'

                                 ,i\_start\_at\_step                => 1

                                 ,i\_concurrency                  => 8

                                 ,i\_max\_rows\_per\_thread          => 500

                                 ,i\_suppress\_emails\_bool         => TRUE

                                 ,i\_halt\_on\_error\_bool           => TRUE

                                 ,i\_master\_monitors\_monitor\_bool => TRUE

                                 ,i\_external\_context\_controller  => '');

-- Add a wait loop here if you want to exit only on migration end.

END;

WAIT LOOP CODE (see above)

--

-- Only necessary if you want to wait until completion.

   WHILE nvl(pkg\_pre\_etl\_tools.fn\_get\_context\_value

(i\_attr => 'MGCTX\_MIGRATION\_RUNNING')

            ,'FALSE') = 'TRUE'

   LOOP

         dbms\_lock.sleep(seconds => 5);

   END LOOP;

Once your migration has completed you may wish to test what the exit state was.

Use: -

BEGIN

/\*------------------------------------------------------------------------------------

\*\* Returns the exit status of the migration

\*\* SUCCESS / FAILED / KILLED / UNKNOWN

\*\*

\*\* UNKNOWN means a migration hasn't just been run or was run before we had a way of

\*\* capturing the exit status.

\*/

IF migration\_run\_framework.fn\_get\_exit\_status IN

('KILLED', 'FAILED')

THEN

raise\_application\_error(-20000

,'Migration ended with status : ' ||

migration\_run\_framework.fn\_get\_exit\_status);

END IF;

END;

## Running a migration - Parameters

There are quite a few parameters to control the migration.

|  |  |
| --- | --- |
| **i\_batch** | **The batch number to run.**  **This is the batch of staging rows you want to import.**  **The staging table MUST have the following 2 columns: -**  **migration\_batch\_id**  **migration\_status** |
|  |  |
| **i\_migration\_group** | **The migration group you want to run.** |
| **i\_exception\_group** | **The group you want to run if it all goes wrong.** |
|  |  |
| **i\_context** | **Migrations need to run under a context.**  **Contexts are a way of configuring a migration so that the behave differently (typically write to other schemas).**  **Contexts are used in conjunction with substitution values. Substitution values are set as ${value} so they are a good way of setting schema names.** |
|  |  |
| **i\_always\_run\_steps\_comma\_sep** | **This is useful if the migration has specific initialisation that should run EVEN IF you are starting from a later step. For example, contexts that need to be setup.**  **If this is NULL, then no steps will be run before starting at i\_start\_at\_step.** |
|  |  |
| **i\_start\_at\_step** | **Start at this specified step in the migration group.**  **The default is 1 (i.e. the beginning).** |
|  |  |
| **i\_start\_at\_thread** | **Under certain circumstances you might want to restart the**  **migration from a specific thread within a step.** |
|  |  |
| **i\_stop\_at\_step** | **Stop at this step. NULL = run till end.**  **The stop step is completed before stopping. In other words, this step is the last to run.** |
|  |  |
| **i\_concurrency** | **The maximum concurrency (only applies to steps that use the driver table ON\_ETL\_DRIVER) - Default is 10. If a step doesn't use the ON\_ETL\_DRIVER then this has no effect.** |
|  |  |
| **i\_max\_rows\_per\_thread** | **This is a target value (sometimes a worker thread might be a little above or below this value). We don't want to have huge amounts of work per thread.**  **This limits threads which use the ON\_ETL\_DRIVER to only processes this number of source rows per commit.** |
|  |  |
| **i\_suppress\_emails\_bool** | **TRUE / FALSE - Should emails be suppressed?** |
|  |  |
| **i\_halt\_on\_error\_bool** | **If TRUE, then the monitor will HALT the migration on the**  **detection of an error.** |
|  |  |
| **i\_master\_monitors\_monitor\_bool** | **If this is TRUE, the master will monitor if the monitor job is**  **running.** |
|  |
|  |  |
| **i\_external\_context\_controller** | **Sometimes you will want to control the behaviour of a migration run based on the setting of an external context. For example, you may which to truncate a target before migrating to it, or you might want to try to keep the target**  **data and add to it. Rather than changing the migration for**  **each run case, use this to set a context value up.**  **Example: TRUNC\_TARGET=TRUE;COMPARISON=TRUE etc... note that**  **context setups are delimited by semi-colon. You can then**  **check for these contexts in your +Process steps.**  **Example:-**  **IF pkg\_pre\_etl\_tools.fn\_get\_context\_value**  **(i\_attr => 'TRUNC\_TARGET') = 'TRUE' THEN**  **execute immediate 'truncate table target\_schema.xyx';**  **END IF;** |
|  |  |
| **i\_job\_class\_name** | **The job class if omitted then the default of DEFAULT\_JOB\_CLASS**  **is used.** |

## How to kill a migration.

If you have started your migration by accident and want it to end prematurely you will need to kill it.

Use the below to do that.  This will also need to be called if you have launched a migration step manually (for example to debug a step - after setting all the appropriate contexts).

BEGIN

   migration\_run\_framework.kill\_launcher(i\_killer            => USER

                                        ,i\_skip\_monitor\_bool => FALSE);

END;

/

## Performance Evaluation

Sometimes mistakes creep into the functions and lookups used in the migration (not errors, but functional mistakes, like not using an index in a query).  You can easily tell where all the time is spent in the JSON creation by looking the introspection tables.

SELECT \*

FROM pre\_etl\_introspection t

WHERE t.monitored\_step LIKE 'MY-MIG%'

/\* The monitored step starts with the Migration Group Name.

See group\_name in select \* from PRE\_ETL\_MIGRATION\_GROUPS t \*/

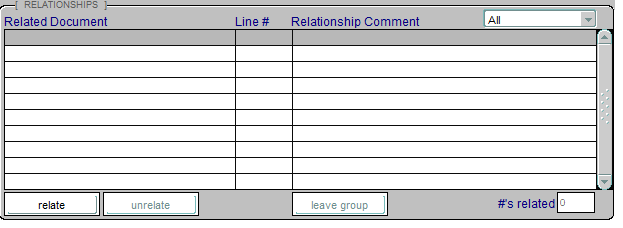
ORDER BY t.monitored\_context

|  |
| --- |
|  |

Look out especially for: execute dynamic select in the MONITORED\_ASPECT: you will see minimum, maximum and average times.  This tells you the amount of SQL execution time. There are lots of other interesting bits of information in this result that will help you tune your migration.

## Relating JSON lines.

It is possible to create connections between documents so that any change on one is reflected on them ALL.  This is handy for fields which are BOTH content in JSON and real first-class Oracle table columns on the same table that will hold that JSON content.



It’s a very EASY MISTAKE to create a relationship and then forget that when you make a change to ANY of the mappings/functions it is changing ALL the related document lines.

If you want to modify a field be sure that it works for all the related ones – IF IT DOESN’T then rather than using the “unrelate” button (which would actually make the highlighted “related document” line in the multi-line table (in the picture above) not have any functions/mapping etc) – **INSTEAD** use the leave group button (all the others will remain related, and ONLY the one you are looking at will be unhooked from the relationship) – you can then make changes that will not affect other related documents.  You can still navigate to other documents and copy any mapping/comment and then use the back button to return to the document you are on to paste in the code/mapping.

Generally, I’d suggest its safer to *not* relate.  Simply copy.  Relationships make *most* sense for content (aka payload) JSON and the table columns in the table that has the payload, although there is nothing stopping you from relating anything to anything.

Furthermore it is suggested, for simplicity whilst getting to know the tools, **NOT** to relate *just* because it’s a comment (i.e. has no mapping) for example for a literal (i.e. a function :  = ‘EXODUS’) – because if someone forgets and suddenly wants to put some other value in there for just their document they might forget about the relationship that exists and they will alter it for EVERYONE.

## Mapping and Map/Rules

By default, Exodus installs with two Map/Rules (MR Groups – not to be confused with Migration Groups):

* DEFAULT
* HISTORICAL

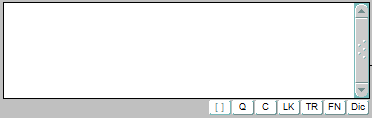
If you require others, then you will need to manually create them by inserting into the PRE\_ETL\_MR\_GROUP table.

cid:image001.png@01D52112.65DF0420

Mappings are made by selecting a target attribute (on the right of the window – these are always some kind of JSON document) – any Target JSON ‘content’ document or TABLE document (tables are still expressed as some JSON), and then selecting the source field (on the left of the window) and pressing the “map” button.

Any mapping may be further enhanced by adding functions or lookups to define the target manipulation required.

Only FN (functions), (LK) Lookups, ([]) Array SQL can accept code, and only then if they start with the correct syntax. (functions must start with an equals sign =, Lookups with a SELECT (or WITH) – (although WITH is not possible in table document mappings, it only usable in a content JSON document – “Stored Document”).



Map/Rules groups allow developers to map and supply rules for the same target tables but for different usages. In other words, let’s assume a developer wants to migrate to a Target Table of MY\_TARGET\_TAB. Let us also assume the source for this table is TAB\_A. The developer can map columns from TAB\_A and supply Function comments and Lookup comments to achieve this. Then, later in the same migration, maybe the developer wants to use TAB\_B as a source for the same target table. The developer has 2 choices, recreate another definition for the target table, or change map/rules and apply the mappings and rules from the new source TAB\_B. The latter is the better choice. Note that it is also possible to use alternate Map/Rules for the content JSON (also known as the payload).

Typically, all mappings take place under “DEFAULT”.

If you need a NEW mapping group let’s say ‘Special Case Mappings’... then you can create one (manually see above).

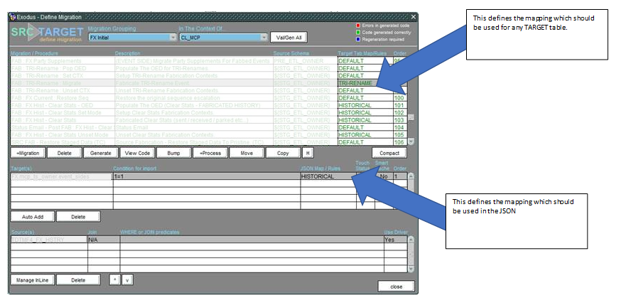
When you choose your mapping group ‘Special Case Mappings’: you are mapping independently from any other mappings (even to target tables/documents that have been mapped before).

## Using the Mappings in a Migration

When you come to define your migration, you can specify how the migration should use the mappings you have made.

Typically, the TABLE and JSON use the same Mapping Group, but not always.

The picture below shows a situation where the TARGET TABLE used a different mapping group from the content JSON, although it should be noted that the data came from the same source.



## Mapping Tips

Whilst performing mapping operations, clicking on the TARGET JSON lines will cause the tool to locate any mapped columns from the SOURCE. Clicking on an unmapped JSON line will highlight the <Not Mapped> (first row in the SOURCE). This can take focus away from a SOURCE column that you’d rather stay focused on.

Use this arrow to refocus back to the previously selected source column.



The image above shows the mapping buttons, and the line number of the JSON TARGET that is being mapped. You may also copy mappings and comments from one TARGET to another TARGET JSON line. Hold down the shift key to skip comment copying whilst clicking the copy button (this will cause only the column mappings to be copied).

Chapter

4

# Configuring Exodus

## Manual Configuration

Some tables need to be manually configured. There is no current facility to manage these tables within the EXODUS GUI.

|  |  |  |
| --- | --- | --- |
| Table Name | Description | Hints |
| pre\_etl\_migration\_groups | A grouping to contain all the migration steps (+Process and +Migration) | Use the display order column to keep frequently accessed groups at the beginning of the list.  DO NOT use underscores in the group\_name. |
| on\_etl\_params | Every Migration Group (see pre\_etl\_migration\_groups above), needs to have its own parameters.  See the table below. | Copy from an earlier migration groups parameters  INSERT INTO on\_etl\_params  SELECT 'NEW-MIG'  ,t.param\_name  ,t.param\_value  ,t.param\_description  FROM on\_etl\_params t  WHERE t.migration\_group = 'MY-MIG' |
| pre\_etl\_mr\_group | mapping / rules groups | If you need to map many sources to the same target tables you can use multiple groups. For example there might be 3 source tables to write to 1 target. The mappings will be different for these. Common cases might be Current Data, Historical Data. |
| pre\_etl\_params | Parameters for the exodus tool. | Currently only 2 params.  INTROSPECTION  ACCESSIBLE\_SCHEMAS  INTROSPECTION can have a value of ON or OFF (this defines if the migration should capture performance characteristics.  ACCESSIBLE\_SCHEMAS a semicolon separated list of schema names that will show their tables in the pick list of tables in the manage JSON window. |
| pre\_etl\_run\_contexts | Context of the run (not to be confused with Oracle global contexts) | Typically, DEV / PROD etc…  This is a heading group for the substitutions below. |
| pre\_etl\_substitution\_values | Values that are constants under a run context. | For example, ${STG\_ETL\_OWNER} could be different depending if you are running as DEV or PROD |

## on\_etl\_params

Every Migration Group will need their own parameters.

|  |  |  |  |
| --- | --- | --- | --- |
| **MIGRATION\_**  **GROUP** | **PARAM\_NAME** | **PARAM\_VALUE** | **PARAM\_DESCRIPTION** |
| MY-MIG | MIG\_EMAIL\_RECIPIENTS\_DEV | [christian.leigh@yourorg.com,someone.else@yourorg.com](mailto:christian.leigh@yourorg.com,someone.else@yourorg.com) | Comma seperated list of email addresses to get development emails. |
| MY-MIG | MIG\_EMAIL\_RECIPIENTS\_MAN | [christian.leigh@yourorg.com,someone.else@yourorg.com](mailto:christian.leigh@yourorg.com,someone.else@yourorg.com) | Comma seperated list of email addresses to get management emails. |
| MY-MIG | MIG\_ETL\_CONSUMER\_GROUP | ETL\_GROUP | Resource group for ETL tasks |
| MY-MIG | MIG\_SMTP\_HOST | mailhost | The email host. |
| MY-MIG | MIG\_STATUS\_EMAIL\_SENDER | [migration@yourorg.com](mailto:migration@yourorg.com) | The email address of the sender. So people can setup rule filters etc. |
| MY-MIG | MIG\_UPDATE\_CYCLE\_MINS\_DEV | 30 | Frequency of email updates for developers and IT staff. |
| MY-MIG | MIG\_UPDATE\_CYCLE\_MINS\_MAN | 120 | Frequency of email updates for managers. |

## pre\_etl\_substitution\_values

Common substitution values are given below.

|  |  |  |  |
| --- | --- | --- | --- |
| **CONTEXT\_NAME** |  | **SUBSTITUTION\_KEY** | **SUBSTITUTION\_VALUE** |
| DEV |  | ${PARALLELISM} | 1 |
| DEV |  | ${STG\_ETL\_OWNER} | DEV\_STG\_ETL\_OWNER |
| DEV |  | ${RUN\_CONTEXT} | DEV |

Chapter

5

# Tips

## Handling Migration Failure

A common requirement might be to recover the target to the original state that existed BEFORE the migration had begun. You can achieve this by defining an external context (**i\_external\_context\_controller** param) and combining that with the **i\_exception\_group** param. Then you should define an early +Process step to do something like the following (I have used an external controller called **BACKUP\_EXISTING\_TARGET\_DATA**) - I then go on to set up another context called **MIG\_BACKUP\_COMPLETED**.

The **MIG\_BACKUP\_COMPLETED** context will be useful to know if the backup had originally succeeded.

This pattern is very useful if your migration can take a long time (for example overnight) - your organization may have a need to have at least some functional data available (even if its not the most recently migrated data) - so you might want to automatically recover the last known "good data" so you do not hold up other work that needs to occur on that database.

### Step 1: Set up an early +Process to capture the original schema tables.

(**IMPORTANT NOTE:** **ONLY THE TABLES DEFINED** AS (Table Document) in the exodus tool will be backed up. **NOT ALL TABLES** in the target schema - if you want ALL tables then they will ALL need to be included into the (Table Document) type tables in the Exodus Tool)

BEGIN

IF pkg\_pre\_etl\_tools.fn\_get\_context\_value(i\_attr => 'BACKUP\_EXISTING\_TARGET\_DATA') =

'TRUE'

THEN

user\_backup\_restore\_steps.pr\_backup\_schema(i\_schema\_name => '${DATA\_CUSTOMER\_OWNER}'

,i\_context => '${RUN\_CONTEXT}'

,i\_overwrite\_bool => FALSE);

user\_backup\_restore\_steps.pr\_backup\_schema(i\_schema\_name => '${DATA\_REFERENCE\_OWNER}'

,i\_context => '${RUN\_CONTEXT}'

,i\_overwrite\_bool => FALSE);

-- We need to know if the backup completed in case we need to run restore.

-- We DON'T want to run the restore if the backup didn't complete.

pkg\_pre\_etl\_tools.pr\_set\_context(i\_attr => 'MIG\_BACKUP\_COMPLETED'

,i\_value => 'Y');

END IF;

END;

**NOTE ABOVE**: ${DATA\_REFERENCE\_OWNER} and ${DATA\_CUSTOMER\_OWNER} are both substitution variable/constants setup in the PRE\_ETL\_SUBSTITUION\_VALUES table.

### Step 2: Create a Migration +Process step late in the migration

As the very last step or after the main body of the migration. This will clean up any backups because they can take up quite a bit of space.

DECLARE

l\_backup\_status VARCHAR2(1) := nvl(pkg\_pre\_etl\_tools.fn\_get\_context\_value(

i\_attr => 'MIG\_BACKUP\_COMPLETED'

,i\_fail\_if\_no\_run\_context\_yn => 'Y')

,'N');

BEGIN

IF l\_backup\_status = 'Y'

THEN

-- Set to cleanup mode (we don't want to fail here and recover after all our hard work.

pkg\_pre\_etl\_tools.pr\_set\_context(i\_attr => 'MIG\_BACKUP\_COMPLETED'

,i\_value => 'C');

user\_backup\_restore\_steps.pr\_remove\_backups(i\_schema\_name => '${DATA\_CUSTOMER\_OWNER}');

user\_backup\_restore\_steps.pr\_remove\_backups(i\_schema\_name => '${DATA\_REFERENCE\_OWNER}');

--

migration\_run\_framework.pr\_log(i\_log\_type => migration\_run\_framework.fn\_info

,i\_log\_entry => 'BACKUPS OF ORIGINAL TARGET DATA CLEANED UP'

,i\_log\_batch => :batch);

END IF;

END;

### Step 3: Create an EXCEPTION Migration Group - this needs to be done manually (once)

You may need to use the R button (refresh button in the main window) to see this if the Exodus interface is already running).

In your IDE of choice connected as PRE\_ETL\_OWNER.

|  |
| --- |
| INSERT INTO pre\_etl\_migration\_groups  (group\_name  ,group\_title  ,group\_description  ,display\_order)  VALUES  ('EXCEPTION GROUP'  ,'Exception Group'  ,'Exception Group To Run In The Event Of Unexpected Failure'  ,900)  /      COMMIT  / |

### Step 4: Create a RESTORE TARGETS (+Process Step) similar to the following…

In the NEW EXCEPTION GROUP (that was created in Step 3)

DECLARE

l\_backup\_status VARCHAR2(1)

:= nvl(pkg\_pre\_etl\_tools.fn\_get\_context\_value(i\_attr => 'MIG\_BACKUP\_COMPLETED'

,i\_fail\_if\_no\_run\_context\_yn => 'Y')

,'N');

BEGIN

IF l\_backup\_status = 'Y'

THEN

user\_backup\_restore\_steps.pr\_restore\_schema(i\_schema\_name => '${DATA\_CUSTOMER\_OWNER}'

,i\_context => '${RUN\_CONTEXT}');

user\_backup\_restore\_steps.pr\_restore\_schema(i\_schema\_name => '${DATA\_REFERENCE\_OWNER}'

,i\_context => '${RUN\_CONTEXT}');

COMMIT;

-- If we have successfully restored we can get rid of the backups.

user\_backup\_restore\_steps.pr\_remove\_backups(i\_schema\_name => '${DATA\_CUSTOMER\_OWNER}');

user\_backup\_restore\_steps.pr\_remove\_backups(i\_schema\_name => '${DATA\_REFERENCE\_OWNER}');

--

migration\_run\_framework.pr\_log(i\_log\_type => migration\_run\_framework.fn\_info

,i\_log\_entry => 'ORIGINAL TARGET DATA RESTORED'

,i\_log\_batch => :batch);

ELSIF l\_backup\_status = 'C' THEN

migration\_run\_framework.pr\_log(i\_log\_type => migration\_run\_framework.fn\_warning

,i\_log\_entry => 'DID NOT RESTORE BECAUSE BACKUP WAS CLEANED UP'

,i\_log\_batch => :batch);

ELSE

migration\_run\_framework.pr\_log(i\_log\_type => migration\_run\_framework.fn\_warning

,i\_log\_entry => 'DID NOT RESTORE BECAUSE BACKUP DID NOT HAPPEN.'

,i\_log\_batch => :batch);

END IF;

END;

## Useful scripts

After making changes to the migration metadata you will almost certainly want to keep those changes safe.  One way to do this is to export the metadata and check the dump into a version control system.  See the below attached scripts.  EXPORT makes a dump and IMPORT is used to restore a dump.  Note that you should be very careful not to restore data over someone else’s changes.

**IMPORTANT:** You should make regular exports and include them with your check-in to the source control system.

Locate these MSDOS batch files in commands.

export\_etl\_owner.cmd

import\_etl\_owner.cmd

## Getting a report of the mappings.

It’s useful to take regular reports of mappings in case as mistake is made and a mapping and its associated function/ lookups etc. become damaged. You should spool the output somewhere and keep it for later lookup.

This is also useful to monitor progress on your mappings. Its suggested that the output is spooled and captured as a source controlled file.

\scripts\mapping\_reports\mapping\_report.sql

## Launching jobs from within the run steps

If you need to launch a job from within a run-step ( a +Process) use the call :-

migration\_run\_framework.pr\_start\_managed\_job(i\_job\_name => 'MY\_JOB\_1'

,i\_job\_sql => 'BEGIN your\_code\_here END;')

That way the migration framework will "KNOW" about your launched job so that it can be stopped and managed correctly through the framework.

## Using DBMS\_PARALLEL\_EXECUTE

If you want to use DBMS\_PARALLEL\_EXECUTE to have some parallel activity that you have specifically crafted, then please use the following 2 calls to notify the run framework.

That way the migration framework will "KNOW" about your DBMS\_PARALLEL\_EXECUTE jobs so that it can be stopped and managed correctly through the framework.

* migration\_run\_framework.pr\_register\_parallel\_task
* migration\_run\_framework.pr\_deregister\_parallel\_task

For example

migration\_run\_framework.pr\_register\_parallel\_task(i\_task\_name => 'MY\_HAND\_CRAFTED\_PARALLEL\_JOB');

dbms\_parallel\_execute.create\_chunks\_by\_number\_col(task\_name => 'MY\_HAND\_CRAFTED\_PARALLEL\_JOB'

,table\_owner => 'XYZ\_SCHEMA'

,table\_name => 'MY\_TABLE'

,table\_column => 'ID'

,chunk\_size => 100000);

dbms\_parallel\_execute.run\_task(task\_name => 'MY\_HAND\_CRAFTED\_PARALLEL\_JOB'

,sql\_stmt => l\_sql\_clob

,language\_flag => dbms\_sql.native

,parallel\_level => l\_threads);

migration\_run\_framework.pr\_deregister\_parallel\_task(i\_task\_name => 'MY\_HAND\_CRAFTED\_PARALLEL\_JOB');

Chapter

6

# SQL\*Loader

We recommend that you export data from any non-Oracle system in a text based flat file using extremely unique multi-character separators to delimit each field (see below). The staged data must have at least one extra column field; “MIGRATION\_BATCH\_ID” defined as a NUMBER. It’s further recommended (but not necessary) that the staged data have an additional column field of “MIGRATION\_STATUS”, defined as a VARCHAR2 (1), although this isn’t necessary for EXODUS to function for the purposes of the migration (but it will come in handy later when you being diagnosing any migration issues). This User Guide cannot be a substitute to familiarizing yourself with the SQL\*Loader documentation.

For example:

@BOR@1@DELIM@Apollo11@DELIM@21.5@DELIM@21/07/1969@DELIM@@EOR@

@BOR@2@DELIM@Apollo12@DELIM@31@DELIM@19/11/1969@DELIM@@EOR@

@BOR@3@DELIM@Apollo13@DELIM@@DELIM@15/04/1970@DELIM@@@EOR@

KEY

|  |  |
| --- | --- |
| @BOR@ | Denotes Beginning Of Record |
| @DELIM@ | Denotes Column Delimiter |
| @EOR@ | Denotes End Of Record |

Once processed through SQL\*Loader the staged table rows for the above might look something like this (**notice the additional fields of** MIGRATION\_BATCH\_ID, and MIGRATION\_STATUS) (You need to create your own stage tables – this can be easily scripted; see PRE\_ETL\_OWNER.PRE\_ETL\_DB2\_COLUMNS):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Surrogate\_ID** | **Mission** | **Hrs On Surface** | **Date** | **MIGRATION\_BATCH\_ID** | **MIGRATION\_STATUS** |
| 1 | Apollo11 | 21.5 | 21/07/1969 | 1 |  |
| 2 | Apollo12 | 31 | 19-11-1969 | 1 |  |
| 3 | Apollo13 |  | 15/04/1970 | 1 |  |

## EXAMPLE Loader control file

OPTIONS

(

DIRECT=TRUE

)

LOAD DATA

INFILE '/data/bulk/source\_file.txt' "str '@EOR@\r\n'"

APPEND

INTO TABLE STAGED\_APOLLO\_DATA

fields terminated by '@DELIM@' TRAILING NULLCOLS

(

BOR FILLER POSITION (1:5)

, SURROGATE\_ID "replace(:SURROGATE\_ID,chr(0),null) "

, MISSION char(8)

, SURFACE\_HOURS "replace(:SURFACE\_HOURS,chr(0),null) "

, EXPECTED\_DATE to\_date(replace(:EXPECTED\_DATE,chr(0),null) , CASE WHEN INSTR(:EXPECTED\_DATE,'/') > 0 THEN 'MM/DD/YYYY' WHEN INSTR(:EXPECTED\_DATE,'-') > 0 then 'YYYY-MM-DD' end )"

, MIGRATION\_BATCH\_ID "1"

, MIGRATION\_STATUS "null"

)

Chapter

7

# Installation Instructions

ORACLE Forms can be fiddly to install, but the payoff is a fantastic tool to perform migration, mapping and other documentation. I won’t describe the steps necessary to install an Oracle Database; the chances are that if you are reading this document then you already are adept at that, or have an Oracle Database already available to host the EXODUS tools. However Oracle Forms can present its own challenges.

These instructions will assume you will be deploying Oracle Forms 11g on Windows10 (64bit).

On your PC create a directory called: C:\InstallFormsAndWeblogic

Note that this directory has no special characters or spaces, and it’s located just off of root on the C: drive. Don’t deviate from this suggestion. You may be lucky in your installation, you might not be. It’s best to avoid the frustrations of doing what might appear at face value to be the “right thing”, but as a result being unable to install the software correctly. These steps also assume a system where no Java has previously been installed. If you have other Java Homes you might need to uninstall them, and then re-install the later.

These steps assume that you are installing on a PC for evaluation purposes, your organization will likely have different standards for installation. It is suggested that you try the deployment in a virtual machine with a virgin installation of Windows 10 (64bit).

## Pre Installation Steps

Locate these files (from Oracle) and download them into that directory:-

* wls1036\_generic.jar
* ofm\_frmrpts\_win\_11.1.2.2.0\_64\_disk1\_1of2.zip
* ofm\_frmrpts\_win\_11.1.2.2.0\_64\_disk1\_2of2.zip
* jdk-7u80-windows-x64.exe

Unpack the zips into the same directory (thereby creating 5 folders):-

* C:\InstallFormsAndWeblogic\Disk1
* C:\InstallFormsAndWeblogic\Disk2
* C:\InstallFormsAndWeblogic\Disk3
* C:\InstallFormsAndWeblogic\Disk4
* C:\InstallFormsAndWeblogic\Disk5

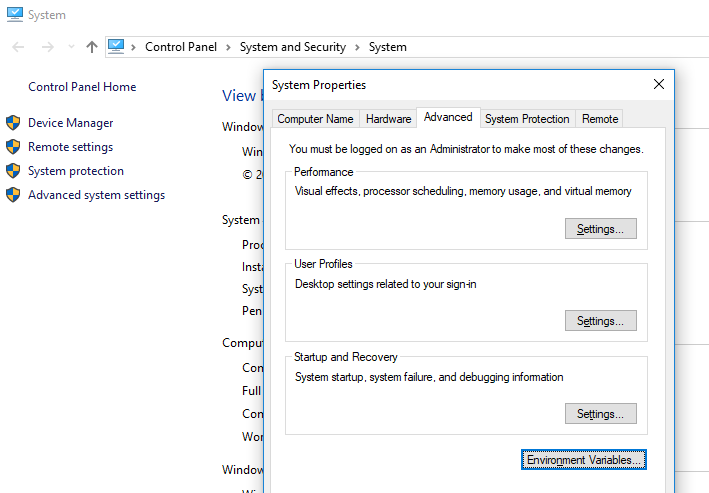
You are now ready to begin the installation.

1. Change these 2 environment variables to have the value: **C:\temp**

**TEMP=C:\temp**

**TMP=C:\temp**

You can find these values in here…



1. Now install the JDK:

Simply double click the file you download during the Pre Installation steps above:-

jdk-7u80-windows-x64.exe

1. Add a new environment variable (located in the same place as described above). (Browse to the location in case its different to the one shown).

**java\_home=C:\Program Files\Java\jdk1.7.0\_80**

1. Modify your path to put the Java bin directory at the very beginning of your path. I.e. your PATH environment variable MUST start with the location of the bin directory of your JAVA installation for example:: **C:\Program Files\Java\jdk1.7.0\_80\bin** (and then a semi-colon separator and the rest of your previous PATH variable).
2. Now open a command prompt.

**cd** to the location you have the installation files.

i.e. C:\InstallFormsAndWeblogic

1. Type : java -jar wls1036\_generic.jar

(press return).

1. Follow the prompts for the installation. When it arrives at the step where it asks about: “I wish to receive security updates via My Oracle Support”. Leave the checkbox ticked. But don’t supply an email address. Then press **NEXT**, and select **YES** in the following 2 dialog boxes.
2. Another window will appear (with a heading of Connection Failed).

Check the box next to the statement: “I wish to remain uninformed of security issues in my configuration or this machine has no Internet access”. Press **CONTINUE**.

1. Leave the options as default. Typical installation. All the way through until the window shows as Installation Complete.

Uncheck the “Run Quickstart” option, and press **DONE**.

1. You may see a warning in the command window about “Could not open/create prefs…”. You can ignore this.
2. Using windows Explorer navigate to the:

C:\InstallFormsAndWeblogic\Disk1

1. Right click on the Setup.exe file and select “**RUN AS ADMINISTRATOR”.**
2. When the install wizard appears; hit **NEXT** and choose the option to Skip Software Updates.
3. Continue through – choosing the default options (until you get to the section for Pre-Requisite checks).
4. When the section for Pre-Requisite checks appears IGNORE the error about system certification (if it appears). Press **CONTINUE**.
5. Chose **Configure for Development** (this is not the DEFAULT) so you must consciously choose that option.
6. In the Create Domain section: Choose a password and make a note of it, as you will need this later.
7. Uncheck the box for “I wish to receive security updates via My Oracle Support”. Press **NEXT** and acknowledge the message about not providing an email address.
8. In the Configure Environment option select ALL the checkboxes.
9. Now chose the defaults all the way through and press INSTALL when the button appears.
10. Now the installation will take place, and it can take a long time, at least 30 Minutes. Make a cup of tea.
11. If the Windows firewall dialog appears: Check the box for “Private Networks, such as my home or work network”. Leave the other checkboxes in that window as they are (probably checked). Press “**ALLOW ACCESS**”
12. You should have had no errors during the installation. If you do something wasn’t configured correctly. Please re-read these instructions.

## Production Systems

For your production system you only need to deploy the PRE\_ETL\_OWNER schema. You don’t need the migration definition tool to actually run a migration, *but* ***you do need*** *the metadata it creates*. You can copy metadata from one PRE\_ETL\_OWNER schema to another. There is a script provided to do this. The script uses the EXP/IMP tools Oracle already provides to achieve this:

export\_metadata\_from\_dev.sh

The export script creates a file called: dev\_meta\_export.dmp

You can import the “.dmp” into another installation of the EXODUS (PRE\_ETL\_OWNER) schema by using:

import\_metadata\_into\_target\_deploy\_db.sh

The import script performs any necessary bootstrapping. For example you might find that you need to run some initializers to create local tables that are required for the migration. Without the import running these “special” steps the migration steps you have defined might not be able to compile. Initializers are actually just other migration groupings that create empty work tables.

Chapter

8

# Installing EXODUS schemas

Instructions can also be found in the readme.txt in the scripts directory.

**NOTE : Errors such as : $'\r': command not found**

**Indicate the files are not in a UNIX format.**

**You may need to : dos2unix \***

1. Run this SQL it MUST return "EXTENDED". If it returns "STANDARD" then you CANNOT deploy this migration suite. You may need to be connected as a privileged user to do this.

**SELECT value FROM v$parameter where name = 'max\_string\_size';**

2. Update the config file BEFORE running this script.

Things to set are...

**deploy\_usr="pre\_etl\_owner" \* Leave this as pre\_etl\_owner.**

**deploy\_pwd="pre\_etl\_owner" \* Whatever password you want pre\_etl\_owner to have.**

**admin\_usr="admin" \* The high privilege account.**

**Could be SYS. (AWS its admin).**

**admin\_pwd="admin1234" \* The password for the high privilege account.**

**admin\_ext="" \* If you are using SYS then this needs**

**to be "AS SYSDBA".**

**stg\_types="DEMO" \* The staging contexts (might just be PROD).**

**Comma separated.**

**temp\_ts="TEMP" \* The Temp TS (most likely left as TEMP).**

**tool\_ts="TS\_MIGTOOL" \* The Tools TS.**

**(whatever TableSpace you have created for the Tool)**

**stg\_ts="TS\_STAGED" \* The staging data TS.**

**(whatever TableSpace you have created for**

**the Staged Data)**

**target\_db="MYDB" \* The db to connect to for deployment. Your DB.**

3. Run the deploy

**. deploy\_master.sh**

4. Deploy the demo Staging schema and Demo Target Schema (see Demo directory)

5. If you are importing the metadata from another system, for

example you are importing the metadata from a development

system. See Note 3 - The import may need to run certain

steps that will require the staging data to be in place.

**. import\_metadata\_into\_target\_deploy\_db.sh**

6. That's it!

To use the web based interface you will need to install Oracle Forms.

Oracle Forms is NOT a free product (it licensed and sold by Oracle), although Oracle do allow evaluation downloads. If your organization already has Oracle Forms you can compile and run the forms distributed with Exodus. You will need to install Oracle forms. Other resources on the internet can be referenced to aid you in this task.