**Migration Manager APIs**

Migration Manager API (available in 7.6.1.3 and Manage) helps move migration packages from a source maximo to target maximo. To facilitate this we have opened up set of rest apis as described below. Note that these apis are designed to handle MM Changes and Snapshot and not migration Collections at this point. Also note that these apis are not designed to define a package – which should be done by the customer SME as part of MM setup. The apis are listed below

**Create a MM package from a package definition.**

This is used to create a package from a package definition.

POST /dm/create

Apikey:<apikey value>

{

 “pkgdefname”:”TESTDD”,

 “readme”: ”moving scripts”

}

The response will have the newly created package json with the **dmpackageid** – which should be used in the next call (/dm/distribute).

The response json will be a representation of the newly created dmpackage – shown below.

{

 “dmpackageid”:..

 “source”:”the source env identifier”,

 “status”:”the status of the package – created/deployed etc – look for the internal value”,

 “filename”:” the name of the file when we download this package”,

 “readme”:”the memo for the created package”,

 “createdate”:”the creation timestamp”,

 “createby”:” the user id that created this package”,

 “pkgdefname”:” the package definition name”,

 “progressstatus”:..

}

**Distribute the newly created MM Package**

This call is used to distribute the package created in the previous step. Distribution is done to a target that is predefined. Targets can be file targets or database targets. In this api, you can choose the target to deploy this package or let the framework choose all possible targets.

POST /dm/distribute

{

 “pkgdefname”:”TESTDD”,

 “**dmpackageid”:31,**

 “targetname”: ”mytarget”

}

Note that the “targetname” property is optional and if not provided, will end up distributing this package to all available targets as defined for this package definition.

The response is the same as the create api, i.e. it’s the representation of modified DMPACKAGE object after the distribution is done. This json representation will also have an additional element “distributionmessage” which is the message that the MM framework generates after a successful distribution.

**Downloading the package**

Once the package is distributed to the target folder, we will need to download the package for uploading it to the target environment. Note that this step is not needed if the target is the target Maximo’s database. In that case, the download and upload steps should be skipped.

GET /dm/download?dmpackagedef= TESTDD&dmpackageid=31

This responds with the package zip. The client code can then upload this zip file to the target Maximo using the next api.

**Uploading the zipped package**

POST <target Maximo api url>/dm/upload?filename=<the downloaded package zip file name>

<<the zipped package>>

This uploads the package to the target Maximo (for File Targets only).

**Deploy the package in the target Maximo**

Next step would be to deploy the uploaded package. Note that we can simply jump to this step if we distributed the package to the target Maximo db in the “distribute” step.

POST /dm/deploy

The successful response would be a json object with a property called “warningmsg” that will have the combined status message as displayed in the classic Maximo UI after a deployment. An error response can also arise for the reasons below:

1. If there is a issue with the package data that fails the target system validation.
2. There is a need for admin mode to be turned on for this deployment.

For both cases the api response would be an error object. For #1, there is no option but to fix the data and recreate the package. But in case of admin mode error (#2), the client should be able to identify the error code (BMXAA0578E) and make the needed calls to setup the admin mode and then call the /dm/completedeploy api and then turn the admin mode off. We will cover those apis below. Note that for #2, the error json will contain the dmpackageid for the failed package.

Below is a sample error json

{

 "Error":{

 "errorattrname": "dmpackageid",

 "extendedError":{

 "moreInfo":{

 "href": "http://localhost:7001/maximo/oslc/error/messages/BMXAA0578E"

 }

 },

 "errattrvalue": “31”,

 "reasonCode": "BMXAA0578E",

 "message": "…..",

 "statusCode": "400"

 }

}

**Turning the admin mode on**

POST /adminmode/on

{

 "numsessions":5

 "logoutmin":1

}

The response should be a 200 OK.

**Checking the state of the admin mode**

GET /adminmode/state

This will return a json containing the state value of 0,1,2. The client code should be looking for a value of 1 – which implies that the admin mode is on. The value of 2 would indicate that its still pending.

**Completing the deployment**

This api would be called in to complete the deployment that got interrupted due to admin mode requirement.

POST /dm/completedeploy

{

 “pkgdefname”:”TESTDD”,

 “**dmpackageid”:<the id returned as part of the error response>**

}

**Turning the admin mode off**

POST /adminmode/off

Turns the admin mode off. The client should do this after the package deployment is complete.

**Reset the event tracking history for Change**

POST /dm/reset

{

 “pkgdefname”:”TESTDD

}

This is a design choice meant to not repeat the same events after they have been migrated.

**Toggle the active flag for Change**

Once a CHANGE package is created and distributed, the event listener gets de-activated. This api is used to reactive that.

POST /dm/toggleactive

{

 “pkgdefname”:”TESTDD

}

**Close the package**

Used for cleanup – after the package content has been donloaded.

POST /dm/close

{

 “pkgdefname”:”TESTDD”,

 “dmpackageid**”:..**

}

Response will the json representation of the DMPACKAGE that got closed.

**Delete the package**

Used for cleanup – after the package content has been downloaded.

POST /dm/delete

{

 “pkgdefname”:”TESTDD”,

 “dmpackageid**”:..**

}