Maximo 76 Scripting Features

Anamitra Bhattacharyya

The Global “service” Variable

The service object is a global object available in all the script points in one form or the other. For most of the script points it's called “service”. As you got through this doc, you will see that the

MIF Object Structure scripts are not directly referring to this variable as “service”. Instead they refer to it as “ctx” (which is nothing but an extended version of the “service” object).

The “service” object helps us make some of the common tasks simple from an automation script. For example the tasks like throwing an error or adding an warning or invoking workflows, invoke channels, logging etc become much easier leveraging the “service” var. And it's not just being easy to use, it's also being the better way to use. For example if you want to throw real time errors, rather than setting the errorkey and errorgrp variables, you should just use the service.error(grp,key) or service.error(grp,key,params) apis to achieve that. It's also leveraged to invoke library scripts as you will see in the next few sections. We will also have examples on how to invoke a MIF Invoke Channel to integrate to some rest apis using the service.invokeChannel(channelname) api.

Library Scripts

Library scripts are good for encapsulating some re-usable logic. In Maximo 76, we can write library scripts as just simple scripts. We can write these scripts in any language and then call from another language.Lets make a library script for making HTTP GET calls. We can make that using a py script like below

```
from psdi.iface.router import HTTPHandler
from java.util import HashMap
from java.util import String

handler = HTTPHandler()
map = HashMap()
map.put("URL",url)
map.put("HTTPMETHOD","GET")
responseBytes = handler.invoke(map,None)
response = String(responseBytes,"utf-8")
```

Here the script is expecting a variable called “uri” to be passed into the script from the invoking script. The response is set to the “response” variable, which can be used by the calling script to get the response data as a string.

A sample script (js) code that can leverage this library script is shown below

```
importPackage(java.util)
importPackage(Packages.psdi.server)

var ctx = new HashMap();
ctx.put("url","http://localhost:7001/maximo/oslc/script/countryapi?_lid=wilson&_lpwd=wilson");
```

```
service.invokeScript("LIB_HTTPCLIENT",ctx);
var jsonResp = ctx.get("response");
var countries = JSON.parse(jsonResp);
```

Note how it creates the ctx and then passes the “uri” to it and then uses “service” variables invokeScript api to invoke the library http script (named LIB_HTTPCLIENT). The script also gets the response from the “response” variable and then uses the JSON.parse(..) api to parse that data to a json object for further processing. We are going to talk about this script more in the lookup script below.

Creating lookups using scripts

Creating lookups can be done using the attribute launch point “Retrieve List” option. As with any attribute launchpoint, we are going to hook the script up with the mbo attribute on which we want the lookup to happen. But just writing the script is not enough for lookups to work. We need to make an entry in the lookups xml for the lookup dialog. We also need to associate the lookup name to the mbo attribute in presentation xml. In the example below we would want to write a lookup for the “address5” attribute in the Address mbo for the multisite application. The address5 maps to a country name using the country code. Say we want to use some of the external rest apis available that provides the list of countries. There are quite of few of those out there. But for this example, just for some fun and learning, let's explore how we can write our own rest apis using Maximo scripting.

Creating REST apis using Automation Scripts

Below we show a script (create this as a vanilla script in autoscript application - no launchpoints) that will do just that.

```
var countries = {"USA" : "United State Of America", "CAN":"Canada", "GBR": "United Kingdom"};
var cc = request.getQueryParam("cc");
var responseBody;
if(cc)
{
  if(countries[cc])
  {
    responseBody = JSON.stringify(countries[cc]);
  }
  else
  {
    responseBody = "";
  }
}
else
{
  responseBody = JSON.stringify(countries);
}
```

We do a lot of json processing here and that is one of the reason why we chose js as the script language. Note the use of 2 implicit variables - request and responseBody. The request refers to the REST api request object (of type com.ibm.tivoli.maximo.oslc.provider.OslcRequest). We leverage that to get http request query parameters for the script to process. The api call will look like

GET /maximo/oslc/script/<script name>

This will return the json with the country code and the country names. You can use any browser or a rest client like chrome POSTMAM to do this test. The api will also optionally support getting the country name based on a country code.

GET /maximo/oslc/script/<script name>?cc=USA

In this example scenario, lets name this script as countryapi. Also make sure that you send in the authentication information as part of this request - like using the maxauth header or for test purpose the _lid=<user>&_lpwd=<password> query params (for maximo native authnetication) or the FORM/Basic auth tokens (for LDAP based auth).

Next we need to create a non-persistent object for storing this country code/country name pairs as maximo lookups only work off of MboSets. We can name the mbo COUNTRY with 2 attributes - name, description.

Next we write the “retrieve list” script.

```
importPackage(java.util)
importPackage(Packages.psdi.server)

var ctx = new HashMap();
ctx.put("url","http://localhost:7001/maximo/oslc/script/countryapi?_lid=wilson&_lpwd=wilson");
service.invokeScript("LIB_HTTPCLIENT",ctx);
var jsonResp = ctx.get("response");
var countries = JSON.parse(jsonResp);

var countriesSet = MXServer.getMXServer().getMboSet("COUNTRY", mbo.getUserInfo());
for(var cnt in countries)
{
   var cntMbo = countriesSet.add();
   cntMbo.setValue("name",cnt);
   cntMbo.setValue("description",countries[cnt]);
}
listMboSet = countriesSet;
srcKeys=["NAME"];
targetKeys=["ADDRESS5"];
```

Note that, we have used the library script LIB_HTTPCLIENT to make the rest api call to get the list of countries. Note the use of the implicit variable “listMboSet” which holds the mboset to be used in the lookup. The target mbo attribute where the value is going to set has a different name (address5) than the src attribute name (“name” in country mbo). So we leverage the implicit vars srcKeys and targetKeys to let the system know where to set the selected value.

MIF (Invoke Channel) Exits Using Scripts

The use case here is to set the city and state in the Organiztion application->Address tab when the user enters the zip code.

Assume we have an external rest api that looks like below

```
GET <zip to city uri>?zips=<zip code>
```

And the response is a json that returns the city and state for the <zip code>. We want to invoke that api when the user enters the zip code and then tabs out (of attribute Address4). To do this, we would need to create an Object Structure for the Address mbo - say named MXADDRESS. We are then going to set-up an invoke channel with an HTTP endpoint that has the url set to the <zip to city uri>. We are going to set the zips query parameter dynamically in the exit scripts. Make sure that you set the “process response” check box and set the request and response Object Structure to MXADDRESS.

We will then create the external exit scripts for both the request and response handling. We will use the Create Scripts for Integration menu option from the Autoscript application to create the request and response exits for Invoke Channel. For both cases, we are going to choose the “External Exit” option. The request (INVOKE.ZIPCHANNEL.EXTEXIT.OUT) exit (in py) will look like below:

```
from java.util import HashMap
from psdi.iface.router import HTTPHandler
from psdi.iface.mic import IntegrationContext
from psdi.iface.mic import MetaDataProperties

epProps = HashMap()
urlProps = HashMap()
zip = irData.getCurrentData("ADDRESS4");
```

```
urlProps.put("zips",zip)
epProps.put(HTTPHandler.HTTPGET_URLPROPS,urlProps)
IntegrationContext.getCurrentContext().setProperty(MetaDataProperties.ENDPOINTPROPS, epProps)
```

The response (INVOKE.ZIPCHANNEL.EXTEXIT.IN) exit (in js) will look like below

```
importPackage(Packages.psdi.iface.mic);

var resp = JSON.parse(erData.getDataAsString());
var irData = new StructureData(messageType, osName, userInfo.getLangCode(), 1, false, true);
irData.setCurrentData("ADDRESS3", resp.zips[0].state);
irData.setCurrentData("ADDRESS2", resp.zips[0].city);
```

Note the response json is assumed to be like this (based on a sample popular rest service out there):

```
{
  “zips”:[
    {
    “state”:”Blah”,
    “city”:”Blah1”
    ….
    }
  ]
}
```

This script could have been optimized a little bit more for the use case given. For example we could have directly set the json data to the Mbo - something that we can do only in case of Invoke Channels.

```
importPackage(Packages.psdi.iface.mic);

var resp = JSON.parse(erData.getDataAsString());
var mbo = IntegrationContext.getCurrentContext().getProperty("targetobject");
mbo.setValue("address3",resp.zips[0].state);
mbo.setValue("address2",resp.zips[0].city);
service.raiseSkipTransaction();
```

Note here how we got the “mbo”. We got it from the IntegrationContext “targetobject” property, which stores the reference of the Mbo to which we are going to set the json response to. Setting the response can be done by MIF using the irData - as we have shown in our previous example. In this example we show how you can do it without needing to create the irData. We use the resp json object to get the state and city and set it directly to the “mbo”. Then we can just skip the rest of MIF processing by asking the service object to skip the transaction. We can use this in other forms of inbound exits - like the user exit and the external exit for Enterprise services. Make sure that you set the data yourself to the mbos that you want and the skip the rest of MIF processing. Also please note that this IntegrationContext “targetobject” property is only available for exits in InvokeChannel.

Next we will write an Attribute LaunchPoint script for the Address4 attribute (zip code attribute in Address object) to invoke this channel when the zip code value is set.

```
service.invokeChannel("zipchannel")
```

As you can see, we use the service variable to do this call. This will call the channel, which in turn goes through the request and response exits.

Maximo Warnings and Errors

Maximo errors in 75 version used to get done by setting these variables - errorgroup, errorkey and params. Setting those did not stop the execution, as the execution will continue till the end of that script and then it will check the flags and throw the error. Often this may not be the expected behavior. To fix this issue, in 76 we added utility methods to the global “service” variable to throw errors and warnings. The example below should replace what you have been doing with errorkey and errorgroups in 75.

```
service.error(“po”,”novendor”)
```

Showing a warning message can be done fairly easily with automation scripts. Say we want to show a warning to the users when they save a PO with no lines.
To do this, first we need to create a warning message from the messages dialog in the db config application. Say we name it "nolines" under the message group "po".

We create an Object Launch Point - Save event on Add and Update. The code below (py) validates the POLINE count and uses the "service" global variable to show the warning.

```
if mbo.getMboSet("POLINE").count()==0 and interactive:
  service.setWarning("po","nolines", None)
```

Interfacing with YN or YNC Dialogs

YNC (aka **Y**es, **N**o, **C**ancel) interactions can now be designed using automation scripts. We need to do some prep work before writing the script. First we need to define a message that support the YNC interaction. This is similar to what we have done in defining the warning message group and key before. Make sure you have the message as informational (I) and support the yes and no buttons.

The use case below launches a yes/no dialog when the priority is set to 1. It asks the user if the he/she wants to set a default vendor for this. If the user select “yes” the script will set the default vendor A0001 and will mark the vendor field as required. If the user selects “no” vendor is marked as not required.

The script shown below associates the var v with vendor and is added as an Attribute Launch point script with “action” event. “assetpr” has been defined in the “asset” group as an informational message with buttons Y and N enabled.

```
def yes():
    global v,v_required
    v = "A0001"
    v_required = True

def no():
    global v,v_required
    v_required = False;

def dflt():
    service.log("dflt")
    params = [str(priority)]
    service.yncerror("asset", "assetpr",params)

cases = {service.YNC_NULL:dflt,service.YNC_YES:yes,service.YNC_NO:no}
if interactive:
   if priority==1:
      x = service.yncuserinput()
      cases[x]()
```

MIF Object Structure Scripts:

Its often required to manipulate the XML/JSON data processing in the Integration Object structure layer. Take the case of outbound messages where the Object structure processing would serialize the Mbo structure into a XML or json message. Some of the common use cases are listed below:

- To override certain values in the xml or json based on the current mbo state.

- Another use case maybe to skip some mbos or attributes based on certain conditions.

You may think that you can do it in exits. However exits are further down the line in outbound processing and may-not be very efficient to let the data get serialized to json/xml and then get dropped. The below example in py shows the 3 functions one can define to interface with the Object structure serialization process. The 3 methods are

Function name	Purpose	
overrideValues(ctx)	Used for the purpose of overriding the mbo attribute values as they get serialized to json/xml	
skipMbo(ctx)	Used for the purpose of skipping the Mbos before serialization. These skipped mbos will not be part of the xml/json generated.	
skipCols(ctx)	Used for skipping mbo attributes before serialization.	

Note that all these functions take in a common parameter ie the ctx - which is an object of type psdi.iface.mos.OSDefnScriptContext. This object contains the state of the serialization and helps the script code to figure out at what level (in the Object structure hierarchy) we are at the point of that call back.

The code below can be applied to the MXPO object structure.

```
def overrideValues(ctx):
  if ctx.getMboName()=='PO' and ctx.getMbo().isNull("description")==True:
    ctx.overrideCol("DESCRIPTION","PO "+ctx.getMbo().getString("ponum"))

def skipMbo(ctx):
  if ctx.getMboName()=='PO':
          if ctx.getMbo().getMboSet("poline").count()==0:
                  ctx.skipTxn()
  elif ctx.getMboName()=='POLINE':
    if ctx.getMbo().isNull("itemnum"):
            ctx.skipMbo()
```

```
def skipCols(ctx):
  if ctx.getMboName()=='POLINE':
        if ctx.getMbo().getBoolean("taxed")!=True:
              ctx.skipCol(['tax1','tax2','tax3','tax4','tax5'])
```

The points to note here are:

- In the override cols note that we are overriding the description only when the object is PO and the description is null. This will not change the PO mbo description. It will just have the description in the json/xml.
- In the skipMbo thing we are will skip the poline processing if line is a service line. We are also going to skip the PO itself if there are no lines for it.
- In the skipCols we check for the mbo in process to be POLINE and if taxed is set to false, we skip all the tax attributes.
- These ctx vars all extend the “service” global var and hence all apis available there can be leveraged here - namely to throw error, call workflow, or to log real time etc.

Note that you can write any combination of these 3 functions in your script code. At least one is required.

To create this script we need to use the action - Create Integration Scripts from the scripting application. We need to choose the Object structure name (in this case MXPO) and the fact that we are using the outbound processing. Next all you need to do is create the script code and save. When you save, the script is auto-magically attached to the Object Structure processing.

In a similar fashion we have a sample for inbound processing. Again as the use case goes - it makes sense to use exit scripting when you want to transform the message payload. But when the message payload is in the format that Maximo consumes and all you want to do is control how the values get set to the Mbo, you should consider using the Object Structure inbound processing scripts. It has a tonne of call-back methods. I am going to only talk about the one I think we are going to use most. I will continue to add the documentation for the other callbacks over the next few months. The example use case deals with setting attribute values in inbound MIF processing in a certain order. Often we see this requirement in the customer implementations where certain mbo attributes needs to get set in a certain order with real time validations. As you already know, MIF inbound would always set attributes in the data dictionary attribute order and will always set it using a DELAYVALIDATION model where the attribute validations and actions are batched up till all the attribute for that mbo has been set. So say you want to set the asset priority and asset type with real time validations and one after the other. This is required - say because these attributes actions need to set some other attributes which are key to subsequent validations. We cannot do that by default in MIF. The known way to do that would be to set these 2 attributes as restricted in the Object Structure. Then we would write an automation script to set these 2 attributes one after the other with real time validations. The code below will do just that.

```
from psdi.mbo import MboConstants

def afterMboData(ctx):
  ctx.log("afterMboData")
  if ctx.getMbo().isNew():
    ctx.getMbo().setValue("priority",ctx.getData().getCurrentDataAsInt("PRIORITY"))
    ctx.getMbo().setValue("assettype",ctx.getData().getCurrentData("ASSETTYPE"))
  else:
ctx.getMbo().setValue("priority",ctx.getData().getCurrentDataAsInt("PRIORITY"),MboConstants.DELA
YVALIDATION)
ctx.getMbo().setValue("assettype",ctx.getData().getCurrentData("ASSETTYPE"),MboConstants.DELAY
VALIDATION)
```

This can be attached to the MXASSET object structure in the same way as described before, the only difference being that this time you would choose the “Inbound Processing” instead of outbound processing.

Note that we used the call back method afterMboData(ctx).
This is specifically used for setting data into the Mbo after the MIF framework has done setting for that Mbo. So when we get the ctx.getMbo(), MIF has already set the attribute values from XML/JSON with DELAYVALIDATION to this mbo. But with DELAYVALIDATION those mbo attributes validations/actions have not yet been invoked. This call-back happens right before that.

We have quite a few such callbacks for inbound processing - each serving its own purpose. Besides these there are script points for rest api actions and rest api queries for Object structure. We are covering those in details in the REST/JSON api documentation.

Event Filters for Publish Channels

Publish Channels provide the outbound integration backbone of Maximo. They can either be event enabled such that when some Mbo’s change their state (ie they get added, updated or deleted) these channels can trap those events and publish a message to the outbound JMS queues. The messages then get picked up from those queues and get pushed to their final destination (using the endpoint/handler framework).
Often we have the need to filter the events such that we want the channel to publish the message only when a certain state condition is met. This used to get achieved by writing event filter classes in java. With 76 this thing can be scripted. To create a Filter script, use the “Script

For Integration" -> "Publish Channel" -> "Event Filter" option in the scripting application. Below is an example of such a filter script on the MXWODETAILInterface publish channel.

```
If service.getMbo().isModified("status") and service.getMbo().getString("status")=="APPR":
    evalresult = False
evalresult = True
```

Note that the evalresult being set to False will indicate the event to be published. So effectively this script will filter all events that did not change the status to APPR for the workorder. You might wonder that this could have been done using a channel rule or channel exit class. That is definitely possible for simplistic rules, although this comes a steep price. The rules get evaluated after the event has been serialized and then passed to the exit layer for evaluation. By that time these rules get evaluated you already paid the price of serialization. So why do that costly process when you can skip it before any processing has been done?
Note that the event filters apply to event based publish channels. Publish channels that are used to export data cannot leverage these filters.
Also note that there is a bug which causes the "mbo" variable to be not accessible directly here. You will have to use the service.getMbo() to get access to the event Mbo here. We will fix this bug in the 7609 release of Maximo.

Before Save, After Save and After Commit Event in Object Launch Point

75 Scripting supported only the "Before Save" events in the Object Launchpoint. Although that is the most common place where we do customizations, we often see the need to attach some customizations at the after save and after commit events - mostly when we are doing some actions that need to be done after the Mbo has been saved or committed. After save is the phase when the Mbo's sql statement (insert/update/delete) has been fired, but the commit has not happened. After commit stage happens when the Mbo in the transaction has been committed ie the database commit has been successfully executed.

After save is events are good for writing to an external storage. For example - Maximo Integration Events are processed at the after save event. In this phase if the write fails, we can always roll back the Mbo transaction. Also at this stage - all Mbo validations have passed and hence the chances of failure due to Maximo business logic failure is very low.

After commit events are good for actions that require your Maximo data to be commited. For example you may want to send an SMS only when an Asset has been reported "BROKEN".

You can access the "mbo" variable at all of the script points here.

Controlling Mbo Duplicate using Scripting

Say we want to hold the reference of Workorder from which we created (copied from) a new workorder. We would create a custom attribute called “copiedfrom” in the Workorder object. We would then need to customize the duplicate process of the Workorder to store the “wonum” of the original workorder in the “copiedfrom” attribute of the newly duplicated mbo. To do this, we will create a vanilla script with the name <Mbo name>.DUPLICATE. For example in this case we will name it WORKORDER.DUPLICATE to intercept the duplicate event. The name <mbo name>.DUPLICATE will intercept the duplicate event for the mbo named <mbo name>. We will have 2 implicit variables - **mbo** and **dupmbo** to leverage in this script. The **mbo** variable will point to the original mbo and the **dupmbo** will point to the newly duplicated mbo. We can now set the “copiedfrom” attribute using the script code below

```
dupmbo.setValue(“copiedfrom”,mbo.getString(“wonum”))
```

This script is called after all duplicate logic is executed in the Mbo framework. Effectively you can write logic in this script using both the mbo and dupmbo to control what gets duplicated as well as do some post duplicate actions (like the sample above).

CanDelete and CanAdd in Object Launch Point

We can control whether we can add or delete a Mbo using scripting Object Launchpoint “Allow Object Deletion” and “Allow Object Creation”.
Can Add
This is an Object Launch Point - "Allow Object Creation" where you can control whether you can add a new Mbo, given the current
state of the system. This point pairs up with the canAdd callback that we have in the MboSet framework.
Lets take a use case where we want to validate that a POLINE can be added to a PO only when the PO has the vendor information set.
The script code to do that is shown below:

```
if mboset.getOwner() is not None and mboset.getOwner().getName()=="PO" and
mboset.getOwner().isNull("vendor"):
  service.error("po","novendor_noline")
```

Note that here, there is no implicit variable called "mbo" as the launch point is invoked before the Mbo is created. At that point
all you have is the MboSet (implicit variable "mboset") for the POLINE.
If you are wondering why this cannot be done using the init Object launch point, the answer is that its little too late. At that point the Mbo has

already been created and added to the set. Rejecting it at the point would have not helped. Also note the usage of the global "service" variable here to throw the error in real time. This effectively replaces setting the errorgrp and
errorkey variables to throw error.

Can Delete
Similar to the "Can Add", this Object Launch Point helps validate whether a Mbo can be deleted or not. Going with the POLINE object, say we want to enforce a validation that the line should not be deleted, if the PO is of priority 1.

```
if mbo.getOwner() is not None and mbo.getOwner().getName()=="PO" and
!mbo.getOwner().isNull("priority") and and mbo.getOwner().getInt("priority")==1:
   service.error("po","nolinedelete_forpriority1")
```

Note that in this case, the mbo is already there (which we are trying to validate for deletion) and hence we can leverage implicit variable mbo to do the validation.

MIF Endpoint Scripts

We can write MIF endpoint handlers using automation script. For example say we want to write a handler for sending emails. This is something that is not there out of the box in Maximo. The steps to do for this would be:

- Use the Add/Modify handlers action from the endpoint application to add a new handler. We are going to name is **scripted** and we are going to set the class name to be com.ibm.tivoli.maximo.script.ScriptRouterHandler.
- We are now going to create a new endpoint for that handler Endpoints app.
- We set the handler to be **scripted.**
- We need to set the “script” property to the name of the script that we are going to write. For example say we name it “emailme”.
- We now got to create a script named emalme.

The script code can be as simple as

```
from psdi.server import MXServer
from java.lang import String

MXServer.getMXServer().sendEMail( to,from,subject, String(requestData))
```

Note in here, we can define the **from** and **to** as literal variables in the script and then set the email addresses there. We can also define another literal variable called **subject** to define a static subject for the email like “Email from Maximo”. We can make it more dynamic and fancy by getting the **to** email from the data etc.

Scripting Cron Tasks in Maximo

Starting 76 we can write scripts for cron tasks too. It follows the same principle as the endpoint handler. We have a cron class - com.ibm.tivoli.maximo.script.ScriptCrontask that helps achieve this. We need to register that class as a new crontask in the crontask application. We then will create an instance of it and set the “scriptname” property to the name of the script that we want this task to run. Next we need to set the schedule and last but not the least - need to define the script. We are then good to go running this newly defined script with the crontasks. It is no different than any other crontasks and you can activate/deactivate the task or change the schedule as you would do in any other cron task.

If you are wondering why we need this when we already have an escalation which is a crontask that works of a given Maximo object. The answer is simple - we often need to schedule jobs that are not just based off Maximo objects. So in those cases, this crontask based scripts would come in handy.

Below we develop a cron script to log an error when the count of maxsessions exceeds a configured limit.

Step 1.
We will first need to register the script crontask definition. For that we will use the Crontask app and the class name would be com.ibm.tivoli.maximo.script.ScriptCrontask. We will also need to create a cron task instance.
In that instance we can set the frequency to 30 sec.
We can set the SCRPTNAME param value to SESSIONCOUNT. This implies that we will create a script names SESSIONCOUNT to do the job.

Step 2.
Next we create the script SESSIONCOUNT. A sample script in py is shown below. Note that the “runasUserInfo” implicit var will hold the userinfo for the cron job.

```
from psdi.server import MXServer
cnt = MXServer.getMXServer().getMboSet("maxsession",runAsUserInfo).count();
if cnt>1:
   service.logError("cnt exceeds 1::"+str(cnt))
```

Step 3.

Next we activate the crontask instance. You need to wait a minute or so for the task to start. You can use another browser instance to login to maximo – just to create another maxsesson record. That is when you can see that log that says "cnt exceeds 1::".

Step 4.
Note that we hard coded the count limit. In this step we can softcode this by leveraging the SCRIPTARG parameter in the Crontask. We can set that to 1. We need to now save and reload the task.

Step 5.
Next we modify the script to leverage that SCRIPTARG parameter using the implicit var called “arg” as below.

```
from psdi.server import MXServer
argval = int(arg)
cnt = MXServer.getMXServer().getMboSet("maxsession",runAsUserInfo).count();
if cnt>argval:
    service.logError("cnt exceeds 1::"+str(cnt))
```

Step 6.
You can now repeat step 3 to see if you get the error log.

Adding validation to the virtual (aka Nonpersistent) mbos (on execute - 7609 feature).

In this sample we are going use the change status dialog from the Asset Application to validate if the memo is filled in when the Asset status is set to BROKEN.

We will write a Object launch point script – MEMOREQD, which will validate on OK button press of the dialog that memo is required for status BROKEN. The event will be “save/add”. This will map to the “execute” call back for Nonpersistent mbos. The object name would be ASCHANGESTATUS. We used Python for this sample.

```
if mboset.getMbo(0).getString("status")=="BROKEN" and
mboset.getMbo(0).isNull("pluscmemo"):
   service.error("memo","reqd")
```

Note the use of the “mboset” variable, which is leveraged to get the current Mbo at 0 index. Now try the change status dialog to see if it throws the memo#reqd error when you do not specify the “memo” for BROKEN.

Leveraging the mbo.add() method in Scripts

It’s a common thing to conditionally set some defaults when a new mbo gets added. We have so far leveraged the Object Launchpoint Init event to do that. We would need to check for onadd flag there to make sure we are adding a new mbo and then set the attribute values. Another way to do that would be to create a script named <mbo name>.ADD. This will automatically add the script to the “add” callback right after the mbo add method execution has happened and right before the init() method gets called. You can use this script to now set default values conditionally. A sample script - ASSET.ADD shows how to default installdate to current date only when the asset type is PUMP.

```
from psdi.server import MXServer
If mbo.isNull(“assettype”) == False and mbo.getString(“assettype”) == “PUMP”:
   mbo.setValue(“installdate”,MXServer.getMXServer().getDate())
```

MMI Script

Maximo Management Interface provides a set of apis (REST/JSON apis) that helps a system administrator gets some insights on the state of the maximo runtime. The root url for that api is

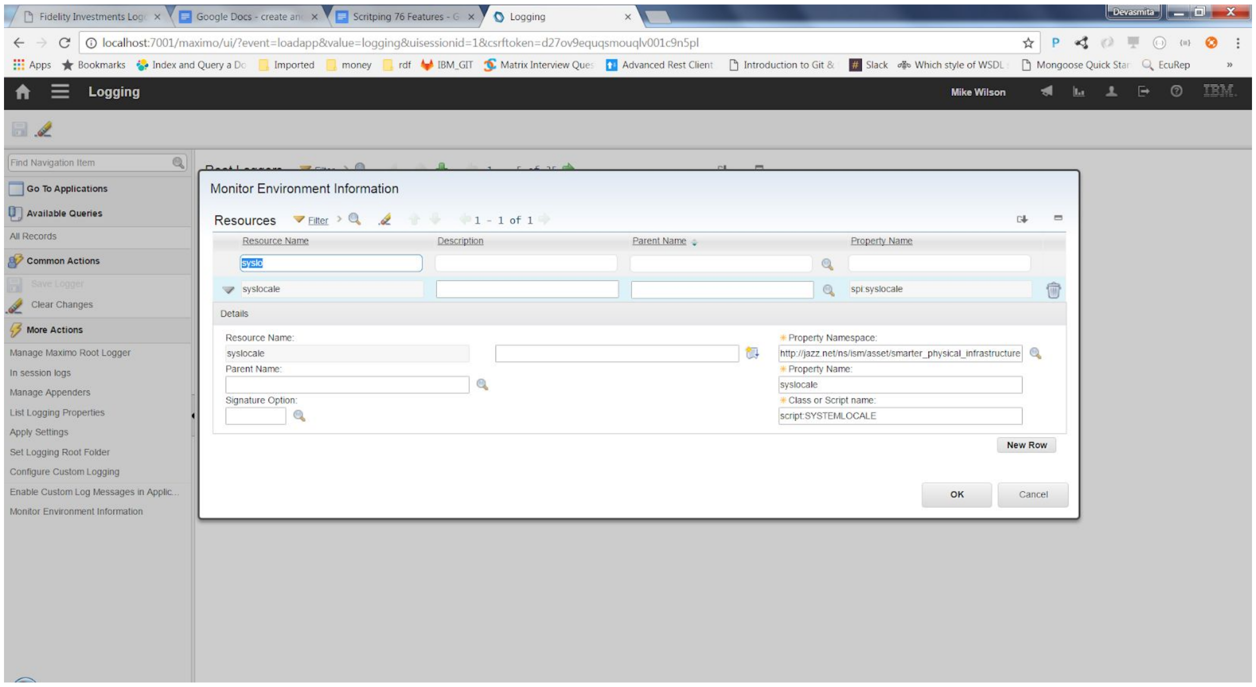
```
GET /maximo/oslc
```

And then you can dive deeper into each server (cluster members) using the /oslc/members api. We provide a bunch of apis that can help you introspect the servers. But of course we do not have all the apis that you will ever need. Fortunately there is a easy way to dynamically add these apis using automation scripts. Below I will provide a sample use case (a real one) that needed us to leverage that feature. Say you need to find out the servers locale as you trying to debug some locale related issue. You can quickly write up a automation script and serve it up as an MMI api using the MMI app - available from the Logging app action in Maximo - Monitor Environment Information. One of the cool aspects of MMI apis being - it gets distributed in every member server and using the MMI api framework - one can invoke these scripted apilets on each server.

The sample script would look like below

```
from java.util import Locale
br.setProperty("syslocale",Locale.getDefault().toString())
```

Lets name it SYSTEMLOCALE. Note the usage of the implict variable “br” that has apis like setProperty(propname,propvalue) and getProperty(propName). The script figures out the system default locale and sets it to the br with a property name of “syslocale”. Now we can add it to the MMI set of apis using the MMI dialog from the logging application.



Resource Name	Description	Parent Name	Property Name
syslo			
syslocale			spi:syslocale

Details

Resource Name: syslocale

Parent Name:

Signature Option:

* Property Namespace: http://jazz.net/ns/ism/asset/smarter_physical_infrastructure

* Property Name: syslocale

* Class or Script name: script:SYSTEMLOCALE

The MMI framework links this script with URL like below

GET /oslc/members/{member id}/systemlocale

Processing JSON data with scripting:

JSON processing can be done natively using the jdk built-in js engine - whether its Rhino or Nashorn. A sample code below shows how to process json from say a rest api invocation using an MIF HTTP endpoint. A sample json that we get by invoking a popular zip code rest api (zippopotam) is shown below

```
{
    post code: "01460",
    country: "United States",
     country abbreviation: "US",
     places:
     [
        {
            place name: "Littleton",
            longitude: "-71.4877",
            state: "Massachusetts",
            state abbreviation: "MA",
            latitude: "42.5401"
        }
      ]
}
```

The js code below shows how we can parse the json above and get the **state** and “**place name**” (city):

```
resp = service.invokeEndpoint("ZIPEP",props,"");
var zipData = JSON.parse(resp);
city = zipData.places[0]["place name"]
state = zipData.places[0]["state abbreviation"]
```

A similar code in python (assuming that we are not using the python json libraries as they dont come by default in the jython jar that we ship) would look a little bit involved as it uses Maximo internal libraries to do the same:

```
from org.python.core.util import StringUtil
from com.ibm.tivoli.maximo.oslc import OslcUtils

resp = service.invokeEndpoint("ZIPEP",props,"")
resp = String(resp)
zipData = OslcUtils.bytesToJSONObject(StringUtil.toBytes(resp))
city = zipData.get("places").get(0).get("place name")
state = zipData.get("places").get(0).get("state abbreviation")
.
```

Invoking workflow from inbound MIF:

This is a common ask - how do I invoke a workflow after my mbo is inserted or updated using MIF. The questions comes up often as the current UI framework allows such a feature to automatically invoke workflow on save of the application mbo. Below is a sample python example that does invoke a workflow that will create an assignment when a new workorder is created using MIF inbound.

To do this we will write an Object Structure script that will invoke the workflow. We will use the new Integration Scripts action from the Scripting application. Select the options – Object Structure -> MXWODETAIL (Object Structure name) ->Inbound Processing.

```
from psdi.server import MXServer
def afterProcess(ctx):
   mbo = ctx.getPrimaryMbo()
   if mbo.isNew():
      MXServer.getMXServer().lookup("WORKFLOW").initiateWorkflow("WOSTATUS",mbo)
```

Note the use of the “ctx” var – which is same as the global implicit var “service” but with added functionality for Object Structure processing.

A simple way to test this would be to use the REST json api:

POST http://host:port/maximo/oslc/os/mxwodetail?lean=1

```
{
“wonum”:”MYLABWO11”,
“siteid”:”BEDFORD”,
“description”:”Invoke workflow test”
}
```

Once we POST this json, it will invoke the script and will create an assignment for the logged in user. You can verify that from the start center assignments portlet.

If interested, you can now enhance the script to make sure that this workflow initiation is only done for transactions that have the status as WAPPR (you can use the ctx.getPrimaryMbo() to check that).

Java 8 and Nashorn engine:

Some of the above example is written using the jdk 7 based rhino js engine. In jdk 1.8, the rhino engine has been replaced with the Nashorn (V8) engine. For example the importPackage command will not work there. You would need to use the JavaImporter function to do the same in Nashorn. You can look into this stackoverflow link for more details on what all changed from Rhino to Nashorn that may impact your script code in js.

http://stackoverflow.com/questions/22502630/switching-from-rhino-to-nashorn