Maximo Business Rules scripting

Maximo® Business Rules (MBR) is a scripting language that provides a cloud-safe way to extend application business logic. MBR, JavaScript and Python are the scripting languages that are available in IBM® Maximo EAM SaaS Flex. MBR is the only scripting language that is available in IBM Maximo EAM SaaS.

A prerequisite for creating scripts using MBR is a full understanding of the principles and practices of automation scripting.

MBR does not support input-output interaction.

This document provides information on the following aspects of MBR:

- Language fundamentals
- Supported operators
- Supported functions
- Supported object attribute access notations
- Script code examples

Language fundamentals

MBR scripts are text files that are made up of expressions. These expressions contain operators, functions, and variables.

When you create an MBR script, you must associate a launch point with the script. The launch point specifies the context that the script runs in. For example, you can configure a script that runs when an attribute is updated. MBR supports the definition of script launch points for the following Maximo artifacts:

- Maximo business objects (MBOs)
- Attributes
- Conditions
- Actions

In MBR script code, you can perform the following tasks:

- Define a function. You must prefix the function name with `:`.
- Create a library script for reusable code. You can invoke a library script from any other script.
- Add a comment. You must prefix the comment with `#`.

In MBR script code, you cannot perform the following tasks:

- Access Java classes directly.
- Call functions or library scripts recursively.
- Define more than one expression in the same line. Expressions are newline delimited.

- Create an expression that spans multiple lines.
- Add inline comments. For example, the following inline comment is not allowed:

  newmbo("opressure","assetmeter") #this is not allowed – put the comment in a new line

In addition, MBR restricts looping capability to the following items:

- Delimited strings, also known as tokens.
- MBOs that are related to the MBO that owns the current artifact. This set of related MBOs is known as an MBOSet.

MBR script code is syntax-validated when you save your script.

Supported operators

Mathematical operators

Operator	Description
+	Addition or concatenation operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Modulo operator
^	Power operator

Boolean operators

Operator	Description
=	Equal
==	Equal
!=	Not equal
<>	Not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
&&	Boolean AND
||	Boolean OR

Note that the *Boolean NOT* operation is implemented by the *not(expression)* function instead of the traditional ! operator.

Supported Functions

Common mathematical functions

Function	Description
not(*expr*)	Boolean negation. Returns 1 if the expression is not 0.
random()	Produces a random number between 0 and 1.
min(*expr1*,*expr2*)	Returns the smaller of two expressions.
max(*expr1*,*expr2*)	Returns the bigger of two expressions.
abs(*expr*)	Returns the absolute value of an expression.
round(*expr*)	Rounds a value by using the current rounding mode.
floor(*expr*)	Rounds a value down to the nearest integer.
ceiling (*expr*)	Rounds a value up to the nearest integer.
log(*expr*)	Returns the natural logarithm of an expression.
log10(*expr*)	Returns the common logarithm of an expression.
sqrt(*expr*)	Returns the square root of an expression.
sin(*expr*)	Returns the trigonometric sine of an angle in degrees.
cos(*expr*)	Returns the trigonometric cosine of an angle in degrees.
tan(*expr*)	Returns the trigonometric tangent of an angle in degrees.
asin(*expr*)	Returns the inverse trigonometric sine of an angle in degrees.
acos(*expr*)	Returns the inverse trigonometric cosine of an angle in degrees.
atan(*expr*)	Returns the inverse trigonometric tangent of an angle in degrees.
sinh(*expr*)	Returns the hyperbolic sine of an angle.
cosh(*expr*)	Returns the hyperbolic cosine of an angle.
tanh(*expr*)	Returns the hyperbolic tangent of an angle.
rad(*expr*)	Converts an angle that is measured in degrees to an approximately equivalent angle that is measured in radians.
deg(*expr*)	Converts an angle that is measured in radians to an approximately equivalent angle that is measured in degrees.
pct(*expr1*,*expr2*)	Returns the equivalent percentage value of *expr1* divided by *expr2*.

Variable setting and variable retrieval functions

Function	Description
setvar(*varname*, *expr[, global]*)	Sets or creates the *varname* variable to have a value of *expr*. The scope of the variable is local by

	default. You can set the *global* parameter to TRUE to make the scope of the variable global.
getvar(*varname*)	Retrieves the value of the *varname* variable. If the variable is defined to have both local and global scopes, the function returns the local value.

Maximo business-specific functions

Function	Description
invokescript(*scriptname*|*functionname*)	Invokes either a library script or a function that is local to the current script. If you invoke a local function, you must prefix the function name with :.
newmbo(*varname*, *relation[, global]*)	Adds a new MBO to the MBOSet for the *relation* relationship. The newly created MBO is visible in the *varname* variable. The default scope of the variable is local. You can set the *global* parameter to TRUE to make the scope of the variable global.
setvaluetombo(*varname*, *attrname, attrval*)	Sets the value of the *attrname* attribute in the MBO that is bound to the *varname* variable. The NOACCESSCHECK flag is set during this update.
setvalue(*attrname*, *attrval*)	Sets the value of the *attrname* attribute in the launch point MBO. The NOACCESSCHECK flag is set during this update.
setvaluenull(*attrname*)	Sets the value of the *attrname* attribute to NULL in the launch point MBO.
setvaluenulltombo(*varname*, *attrname*)	Sets the value of the *attrname* attribute to NULL in the MBO that is bound to the *varname* variable.
error(*errgrp, errkey*)	Creates an MXException error that is identified by the *errgrp* group and *errkey* key.
warning(*wnggrp, wngkey*)	Creates an MXException warning that is identified by the *wnggrp* group and *wngkey* key.
setrequired(*attrname, bool*)	Makes the *attrname* attribute mandatory or optional. You can set *bool* to 1 for mandatory or 0 for optional.
setreadonly(*attrname, bool*)	Makes the *attrname* attribute read-only or editable. You can set *bool* to 1 for read-only or 0 for editable.
sethidden(*attrname, bool*)	Makes the *attrname* attribute hidden or visible. You can set *bool* to 1 for hidden or 0 for visible.
deletethismbo(*varname*)	Deletes the MBO that is bound to the *varname* variable.
deleteall(*relation*)	Deletes all MBOs from the MBOSet for the *relation* relationship in the launch point MBO.

deleteallfromthismbo(*varname*, *relation*)	Deletes all MBOs from the MBOSet for the *relation* relationship in the MBO that is bound to the *varname* variable.
isnullf(*expr*)	Returns TRUE if the expression evaluates to NULL.
invokeworkflow(*wfname*)	Invokes the *wfname* workflow synchronously.
maxprop(*propname*)	Returns the equivalent string value of the *propname* system property.
maxcond(*condname*)	Evaluates the *condname* system condition and returns either TRUE or FALSE.
tobeadded()	Returns TRUE if the launch point MBO is marked for creation.
tobesaved()	Returns TRUE if the launch point MBO is marked for saving.
tobedeleted()	Returns TRUE if the launch point MBO is marked for deletion.
setevalresult(*TRUE/FALSE*)	Sets the Boolean value of the *evalresult* implicit variable in scripts that have a condition launch point.
setthisattrvalue(*value*)	Initializes the value of the launch point attribute in scripts that have an attribute launch point.
scriptvar(*varname*)	Returns the value of the *varname* implicit variable. You must specify *varname* within quotation marks. This function returns values for only the following implicit variables: • *app* • *action* • *interactive* • *launchPoint* • *mboname* • *onadd* • *ondelete* • *onupdate* • *scriptName* • *user*
yncerror(*msggrp, msgkey*)	Displays a message box that is identified by the *msggrp* group and the *msgkey* key. You must configure the message box to contain buttons such as Yes, No, Cancel, and OK.
yncuserinput()	The numeric value that this function returns depends on whether a message box is displayed to the user when this function is called. If a message box is displayed, this function returns a value that indicates which one of the following buttons the user clicks: • Yes

	• No • Cancel • OK If a message box is not displayed, this function returns a default value. To identify the value that this function returns, you can use the isyncyes(), isyncno(), isynccancel(), isyncok() and isyncdflt() functions.
isyncyes(*userinputvalue*)	Returns TRUE if the *userinputvalue* value indicates that the user clicked a Yes button.
isyncno(*userinputvalue*)	Returns TRUE if the *userinputvalue* value indicates that the user clicked a No button.
isynccancel(*userinputvalue*)	Returns TRUE if the *userinputvalue* value indicates that the user clicked a Cancel button.
isyncok(*userinputvalue*)	Returns TRUE if the *userinputvalue* value indicates that the user clicked an OK button.
isyncdflt(*userinputvalue*)	Returns TRUE if the *userinputvalue* value is a default value.

MBOSet aggregation functions

Function	**Description**
countf(*relation[, dateattr, duration]*)	Returns the total number of objects in the MBOSet for the *relation* relationship. You can filter the set of objects that are counted by specifying a date attribute name and a duration. An object is included if the *dateattr* date is between the current date and a *duration* period of time before the current date.
avgf(*relation, attrname[, dateattr, duration]*)	Returns the average value of the *attrname* attribute for all objects in the MBOSet for the *relation* relationship. You can filter the set of objects that are counted by specifying a date attribute name and a duration. An object is included if the *dateattr* date is between the current date and a *duration* period of time before the current date.
maxf(*relation, attrname[, dateattr, duration]*)	Returns the maximum value of the *attrname* attribute for all objects in the MBOSet for the *relation* relationship. You can filter the set of objects that are counted by specifying a date attribute name and a duration. An object is included if the *dateattr* date is between the current date and a *duration* period of time before the current date.

minf(*relation, attrname[, dateattr, duration]*)	Returns the minimum value of the *attrname* attribute for all objects in the MBOSet for the *relation* relationship. You can filter the set of objects that are counted by specifying a date attribute name and a duration. An object is included if the *dateattr* date is between the current date and a *duration* period of time before the current date.

Control flow functions

Function	Description
if(*condition*, *valueiftrue[*, *valueiffalse]*)	Returns the *valueiftrue* value if the *condition* condition evaluates to TRUE. You can specify a value that is returned when the *condition* condition evaluates to FALSE by populating the *valueiffalse* parameter.
foreachmbo(*relname*, *scriptname*|*functionname* [, *cond*])	Runs a library script or local function for each MBO in the MBOSet for the *relation* relationship. The number of iterations cannot exceed the maximum fetch limit that is configured in the application.
foreachtoken(*str*, *splitchar*, *scriptname*|*functionname*)	Runs a library script or local function for each token in the *str* string. The *str* string is delimited into tokens by the *splitchar* delimiter.
continue()	Continues to the next iteration in the loop.
break()	Breaks from the loop.

String manipulation functions

Function	Description
concat(*str1*, *str2*)	Concatenates two strings.
tolower(*str*)	Converts a string to lowercase.
toupper(*str*)	Converts a string to uppercase.
startswith(*str1, str2*)	Returns TRUE when the *str1* string starts with the *str2* string.
endswith(*str1*, *str2*)	Returns TRUE when the *str1* string ends with the *str2* string.
substring(*str, startofffset[, endoffset]*)	Returns a substring of the *str* string starting at the *startofffset* offset. If you do not specify an end offset in the *endoffset* position, the function returns the substring from the *startofffset* offset to the end of the string.
tokenat(*str, splitchar, offset)*	Returns the token at the *offset* offset for the string *str* that is delimited by the *splitchar* delimiter.

Date manipulation functions

Function	Description
now()	Returns the current date and time.
duration(*years, months, days, hours, mins, secs*)	Returns an amount of time specified in years, months, days, hours, minutes, and seconds. You can specify a duration that you can add to or subtract from a date.
date(*year, mon, day*)	Creates a date object for a specific year, month, and day.
datetime(*year, mon, day, hh, mm, ss*)	Creates a date object for a specific year, month, day, and time. You must use two digits to specify each of the hours, minutes, and seconds parameters.

Other utility functions

Function	Description
nvl(*expr1, expr2*)	Returns the *expr1* expression if the *expr1* expression is not NULL. If the *expr1* expression is NULL, this function returns the *expr2* expression.
number(*str*)	Converts a string to a number.
str(*num*)	Converts a number to a string.

Boolean primitive functions

Token	Description
TRUE	Returns a Boolean TRUE value.
FALSE	Returns a Boolean FALSE value.

Supported object attribute access notations

You can access MBO attributes directly in your MBR script code by using the following notations:

Notation	Description
relation1$relation2$....relationN$attr	Returns the value of an attribute in a related MBO. You can traverse several relationships to find the MBO that contains the attribute value that you want.
owner1$owner2$....ownerN$attr	Returns the value of an attribute in the launch point MBO.
modified$attr	Returns TRUE if the *attr* attribute is modified. Returns FALSE if the *attr* attribute is not modified.
prev$attr	Returns the initial value of an attribute.

internal$attr	Returns the internal value of an attribute that is bound to a synonym domain.

Script code examples

Example 1: Adding new MBOs to an MBOSet

Use case: When a new asset is created that has an asset type of GASENG, add two meters, O-PRESSUR and IN-PRESSUR, to the meter MBOSet of the asset.

To create this script:

1. In the Automation Scripts application, create a script that has an object launch point.
2. In step 1 of the wizard, specify the launch point name and select the ASSET object.
3. In the Events section, select the **Save** radio button.
4. In the Save section, check the **Add** check box and select the **Before Save** radio button.
5. In the Script section, select the **New** radio button.
6. In step 2 of the wizard, specify the script name and set the log level to debug.
7. In step 3 of the wizard, paste in the following source code and then click **Create**:

```
#my first MBR code for asset
setvar("isgaseng",(assettype == "GASENG" && countf("assetmeter")==0))
if(getvar("isgaseng"),invokescript(":createmeters"))

#my MBR function that will add the two meter MBOs to the asset MBO
:createmeters
newmbo("opressure","assetmeter")
setvaluetombo("opressure","metername","O-PRESSUR")
newmbo("inpressure","assetmeter")
setvaluetombo("inpressure","metername","IN-PRESSUR")
```

Code analysis

This script contains two lines of main code and a function. The script is run when a user saves changes to an asset record.

In the main code, the following logic is implemented:

1. Create a local variable and set the variable to TRUE if both of the following conditions are true:
    - The asset type is GASENG.
    - The meter MBOSet of the asset is empty.

2. Evaluate the local variable. If the local variable is TRUE, call a function to add MBOs to the meter MBOSet of the asset.

In the function, which is called *createmeters*, the following logic is implemented:

1. Add an MBO to the meter MBOSet of the asset.
2. Set the name of the MBO to O-PRESSUR.
3. Add a second MBO to the meter MBOSet of the asset.
4. Set the name of the second MBO to IN-PRESSUR.

The script uses two internal variables, *assettype* and *assetmeter*, to derive the asset type and the contents of the meter MBOSet of the asset.

Because this script runs when a user saves their changes, the application transaction framework saves and commits the newly created MBOs as part of the main transaction.

You can move the *createmeters* function code into a library script. Remember that library script names do not contain **:** characters. For example, you can move the *createmeters* function code to a library script that is also called *createmeters* and then call the library script in the following statement:

```
if(getvar("isgaseng"),invokescript("createmeters"))
```

Example 2: Validating attribute values

Use case: If an asset has a type of GASENG, ensure that the purchase price of the asset cannot contain a value that is greater than 200.

To create this script:

1. In the Automation Scripts application, create a script that has an attribute launch point.
2. In step 1 of the wizard, specify the launch point name, and select the ASSET object and the PURCHASEPRICE attribute.
3. In the Events section, select the **Validate** radio button.
4. In the Script section, select the **New** radio button.
5. In step 2 of the wizard, specify the script name and set the log level to debug.
6. In step 3 of the wizard, paste in the following source code and then click **Create**:

```
if(not(isnullf(assettype)) && assettype=="GASENG" && not(isnullf(purchaseprice)) &&
purchaseprice>200,error("asset","toomuchcost"))
```

Code analysis

This single-line script is run when the purchase price of an asset is modified.

The script creates an error if all of the following conditions are true:

- The asset type is valid.
- The asset has a type of GASENG.
- The purchase price of the asset has a value.

- The purchase price contains a value that is greater than 200.

In addition to using the internal variables *assettype* and *assetmeter* that are used in Example 1, this script uses the internal variable *purchaseprice* to derive the purchase price of the asset.

This script also uses a sample error group and error key. When you write script code that creates an error, ensure that the message group and message key that you use exist in the error message database.

Example 3: Manipulating dates

Use case: If an asset has a type of BUS, perform the following actions:

- Set the end-of-life date to the install date plus one year. If the asset does not have an install date, set the end-of-life date to the current date plus one year.
- Make the priority attribute mandatory.

To create this script:

1. In the Automation Scripts application, create a script that has an attribute launch point.
2. In step 1 of the wizard, specify the launch point name, and select the ASSET object and the ASSETTYPE attribute.
3. In the Events section, select the **Validate** radio button.
4. In the Script section, select the **New** radio button.
5. In step 2 of the wizard, specify the script name and set the log level to debug.
6. In step 3 of the wizard, paste in the following source code and then click **Create**:

```
if(assettype=="BUS",setvalue("estendoflife",nvl(installdate,now())+duration(1,0,0,0,0,0)))

if(assettype=="BUS",setrequired("priority",TRUE),setrequired("priority",FALSE))
```

Code analysis

This script is run when the type of an asset is modified.

In the script, the following logic is implemented:

- If the asset has a type of BUS, set the end-of-life date:
    - Evaluate the install date of the asset.
    - If the install date is empty, set the end-of-life date to the current date plus one year.
    - If the install date is populated, set the end-of-life date to the install date plus one year.
- If the asset has a type of BUS, make the priority attribute mandatory. If the asset does not have a type of BUS, make the priority attribute optional.

For both of these actions, the script checks to see if the asset has a type of BUS, which is inefficient. You can improve the script by defining a function, moving both actions into the function, and calling the function if the asset has a type of BUS. The following code demonstrates this improvement:

```
if(assettype=="BUS",invokescript(":seteol"),setrequired("priority",FALSE))
```

```
:seteol

setvalue("estendoflife",nvl(installdate,now())+duration(1,0,0,0,0,0))

setrequired("priority",TRUE)
```

The script uses the internal variables *assettype, estendoflife,* and *installdate* to derive the values of the type, end-of-life date, and install date of the asset.

Example 4: Looping

Use case: Two types of meter, out-pressure and in-pressure, are defined for an asset. When a reading is recorded for an asset meter, set the priority of the asset to the numeric difference between the out-pressure and in-pressure meter readings.

To create this script:

1. In the Automation Scripts application, create a script that has an object launch point.
2. In step 1 of the wizard, specify the launch point name and select the ASSET object.
3. In the Events section, select the **Save** radio button.
4. In the Save section, check the **Add** check box and select the **Before Save** radio button.
5. In the Script section, select the **New** radio button.
6. In step 2 of the wizard, specify the script name and set the log level to debug.
7. In step 3 of the wizard, paste in the following source code and then click **Create**:

```
foreachmbo("ACTIVEASSETMETER","":pressurediff")

setvalue("priority",getvar("op")-getvar("ip"))


:pressurediff

if(metername=="O-PRESSUR",setvar("op",number(newreading),TRUE))

if(metername=="IN-PRESSUR",setvar("ip",number(newreading),TRUE))
```

Code analysis

This script contains two lines of main code and a function. The script is run when a user saves changes to an asset record.

In the main code, the following logic is implemented:

- For each type of meter that is defined for an asset, call a function that records the meter reading.
- Set the priority of the asset to the numeric difference between the out-pressure and in-pressure meter readings.

In the function, the following logic is implemented:

- If the function is called for the out-pressure meter, record the out-pressure meter reading in a variable that has a global scope.
- If the function is called for the in-pressure meter, record the in-pressure meter reading in a variable that has a global scope.

Because the meter readings are assigned to variables that have a global scope, the main code can use these meter readings to set the priority value on the asset.

The script uses the internal variables *priority, metername,* and *newreading* to derive the values of the priority of the asset and the name and reading of the meter. The main code uses the internal relation name *ACTIVEASSETMETER* to determine the meter MBOSet for the asset that the loop iterates through.

Example 5: Setting attributes on duplicate objects

Use case: When a work order is duplicated, copy the original work order number into a custom attribute in the duplicate work order.

To create this script:

1. In the Automation Scripts application, create a script that does not have a launch point.
2. In the wizard, specify a script name of WORKORDER.DUPLICATE.
3. Specify MBR as the script language.
4. Paste in the following source code and then click **Create**:

```
setvaluetombo("dupmbo","copiedfrom", wonum)
```

Code analysis

The script name ensures that this script is run when a work order is duplicated.

The script copies the original work order number into a custom attribute in the duplicate work order.

The script uses the internal variables *wonum, dupmbo,* and *copiedfrom* to derive the values of the original work order number, the duplicate work order number, and the custom attribute on the duplicate work order.

Example 6: Using message boxes that include Yes, No, and Cancel buttons

Use case: If the user sets the priority of an asset to a certain value, display a message box that contains Yes and No buttons and asks the user to perform one of the following actions:

- Click Yes if they want a default value to be populated for the asset's vendor attribute.
- Click No if they want the vendor attribute to be optional.

Update the vendor attribute based on which button the user clicks.

To create this script:

1. In the Automation Scripts application, create a script that has an attribute launch point.
2. In step 1 of the wizard, specify the launch point name, and select the ASSET object and the PRIORITY attribute.
3. In the Events section, select the **Validate** radio button.
4. In the Script section, select the **New** radio button.
5. In step 2 of the wizard, specify the script name and set the log level to debug.
6. In step 3 of the wizard, paste in the following source code and then click **Create**:

```
if(priority==1,invokescript(":handle_ync"))

:handle_ync
if(isyncdflt(yncuserinput()),invokescript(":dflt"),if(isyncyes(yncuserinput()),invokescript(":yes"),if(isyncn
o(yncuserinput()),invokescript(":no"))))

:yes
setvalue("vendor","A0001")

:no
setrequired("vendor",FALSE)

:dflt
yncerror("asset", "assetpriority")
```

Code analysis

This script is run when the priority of an asset is modified. The script contains one line of main code and four functions.

In the main code, the following logic is implemented:

- If the user sets the asset priority to 1, call the *handle_ync* function. This function calls other functions to display the message box and perform the action that is indicated by the user.

In the *handle_ync* function, the following logic is implemented:

1. Call the *dflt* function to display the message box that asks the user to perform one of the following actions:
    - Click Yes if they want a default value to be populated for the vendor attribute.
    - Click No if they want the vendor attribute to be optional.
2. Perform one of the following actions:
    - If the user clicks Yes, call the *yes* function to set a default value for the vendor attribute.
    - If the user clicks No, call the *no* function to make the vendor attribute optional.

In the *dflt* function, the following logic is implemented:

- Call the built-in function *yncerror()* to display the message box.

In the *yes* function, the following logic is implemented:

- Set the value of the vendor attribute to A0001.

In the *no* function, the following logic is implemented:

- Set the REQUIRED property of the vendor attribute to false.

This script uses the built-in function *yncuserinput()* to return a value that indicates which button the user clicks in a message box. Note that if you call this function when a message box is not displayed, the function returns a default value.

This script uses the internal variables *priority* and *vendor* to derive the values of the priority and vendor of the asset. In addition, the script uses the internal message group *asset* and message key *assetpriority* to identify the message box that is displayed to the user.