

Decision Management Community Challenge July 2025

IBM ODM Solution by Andy Mac

<https://www.ibm.com/products/operational-decision-manager>

Rules with Regular Expressions



A university is developing a system that helps a student to select certain courses to receive a desired degree. Each course has a unique code, such as “MA523” or “CS38251”. Help a university create a simple decision service that evaluates a set of all courses to split it into two lists of allowed and not allowed courses, following these rules:

RULE 1. If the Degree Code is “MA Single” and the Course Code is similar to MA52# or MA62#, where # stands for any single character, add this course to the list of allowed courses.

RULE 2. If the Degree Code is not “CS%” and the Course Code is similar to CS%288% or CS%289%, where % stands for any combination of characters, add this course to the list of not allowed courses.

Solution Approach

My standard approach to building rule projects with IBM ODM is to :

- Start simple
- Make it as easy as possible for non-technical users to maintain the rules
- Match the requirements as closely as possible
- Enable future improvement/extensions to the rules

Building a dedicated data model for the inputs and outputs is usually the best approach for an ODM rule project as it enables a lot of control and finesse but this requirement is so simple that native types have been used. These are the variables that were created for the solution. I like to add suffixes to the internal variable identifiers to make it clear if they are inputs or outputs to the rule operations. You can also create parameters that are both inputs and outputs.

Variable Set: Variables

Name	Type	Verbalization
degreeCode_in	java.lang.String	DegreeCode
courseList_in	java.lang.String[]	CourseList
allowedCourses_out	java.util.Vector	AllowedList
notAllowedCourses_out	java.util.Vector	NotAllowedList

Writing the rules as close as possible to the original requirement using the default ODM rule language syntax gives the following two rules:

```
definitions
  set 'Course' to a string in CourseList ;
if
  DegreeCode is "MA Single"
  and there is at least one string in { "MA52#" , "MA62#" }
    where Course is similar to this string ,
then
  add Course to AllowedList ;

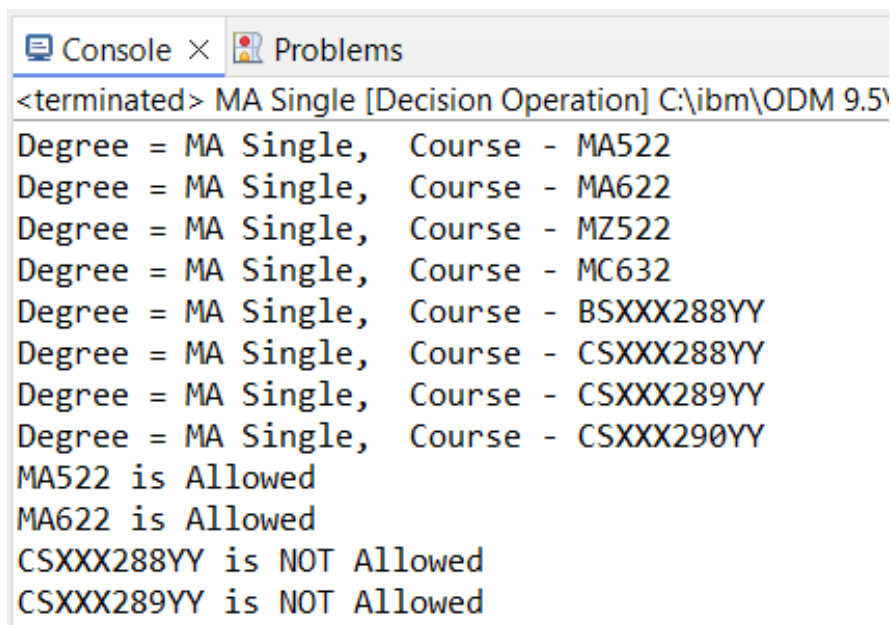
definitions
  set 'Course' to a string in CourseList ;
if
  DegreeCode is not similar to "CS%"
  and there is at least one string in { "CS%288%" , "CS%289%" }
    where Course is similar to this string ,
then
  add Course to NotAllowedList ;
```

As we did not create a specific data model there are no automatic phrases for adding items to a list so I created a couple of virtual "helper" methods for the list addition and also for the customised regular expression handling. These are described in more detail later on.

Execution

The ODM Rule Designer enables you to run different test scenarios immediately without any deployment or code generation steps. Multiple Run Configurations can be created to quickly test different scenarios.

I added a third rule to print the inputs so the logic can be validated very quickly. Here is a screenshot of such an execution:

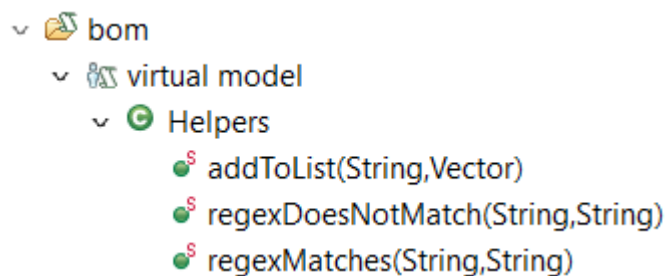


```
Console × Problems
<terminated> MA Single [Decision Operation] C:\ibm\ODM 9.5\
Degree = MA Single, Course - MA522
Degree = MA Single, Course - MA622
Degree = MA Single, Course - MZ522
Degree = MA Single, Course - MC632
Degree = MA Single, Course - BSXXX288YY
Degree = MA Single, Course - CSXXX288YY
Degree = MA Single, Course - CSXXX289YY
Degree = MA Single, Course - CSXXX290YY
MA522 is Allowed
MA622 is Allowed
CSXXX288YY is NOT Allowed
CSXXX289YY is NOT Allowed
```

The definition section of both rules causes the if condition to be evaluated for every item in the CourseList array input parameter. The valid and non valid courses are added to the two output lists.

Helper Methods

The add to list and regular expression tester phrases were manually added to a BOM entry. It is virtual as there is no underlying Java class source code.



Looking at the **addToList** method first.

Member addToList (class: Helpers)

General Information
Name:
Type:
Class:
☒ Static ☐ Final

Member Verbalization

☒ [Remove](#) the verbalization.

☒ [Create](#) an action phrase.

Action : "add <course> to <course list>"
Template:

You specify what the argument types are:

Arguments
Edit the arguments of this member.

Name	Type
course	java.lang.String
list	java.util.Vector

And then the most powerful bit is you can define the business user "verbalization" which specifies how the phrase will appear in the rule language. So you can make this really readable for the business users.

▼ Action : "add <course> to <course list>"

Template:

The user defined angle bracket text strings are optional but if provided they will be shown in the rule editor. These guide the business user as to what kind of data the phrase requires e.g.:

When this phrase is used in a rule you need to specify what code is executed and that is added in the BOM to XOM panel:

BOM to XOM Mapping
Edit the mapping between this BOM member and the XOM.

☒ [Edit](#) the imports.

Body (in ARL)

```
1 list.add(course);  
2
```

For the regular expression phrase I tried to make this match the original requirement as closely as possible:

Member regexMatches (class: Helpers)

General Information

Name:
Type:
Class:
☒ Static ☐ Final

Member Verbalization

✗ [Remove](#) the verbalization.

✚ [Create](#) a navigation phrase.

▼ Navigation : "a string is similar to a string"

Template:

Which looks like the following in a rule:

```
if
    Course is similar to "MA52#"
```

Note that this uses the symbols from the requirements ('#') rather than the default regex matching codes in Java. To achieve this I simply replace the provided symbols by the ones expected by Regex. That is another benefit of using helper methods as it makes such refinements very easy.

```
1 expression = expression.replace("%", "[A-Z]+");
2 expression = expression.replace("#", ".");
3 return text.matches(expression);
```

Note that here I am only allowing uppercase letters for the '%' wildcard but you have the full power of Regex available here and it could be mixed case letters and number etc. The '.' matches any single character but you could restrict this to just letters/numbers etc.

Publishing as a callable ODM Service

To deploy the rules as a callable ODM service you first need to create a Decision Operation as this is where you define the inputs and outputs and name of the service

- ▼ deployment
 - > Deploy both
 - > Filter Courses Operation

virtual model

Filter Courses Operation ×

Decision Operation Signature - Filter Courses Operation

Eligible variables

Select the ruleset variables that you want to use as parameters for the decision operation. Ruleset variables are defined in variable sets.

Input Parameters

Define the parameters required to call the execution.

Parameter name	Verbalization	Type
degreeCode_in	DegreeCode	java.lang.String
courseList_in	CourseList	java.lang.String[]

Output Parameters

Define the parameters that are initialized and returned by the execution.

Parameter name	Verbalization	Type
allowedCourses_out	AllowedList	java.util.Vector
notAllowedCourses_out	NotAllowedList	java.util.Vector

And this is what a successful call looks like in an API testing tool:

The screenshot shows an API testing tool interface. The top bar indicates a POST request to `http://localhost:9090/DecisionService/rest/v1/Utilities/Filter_Courses/OPENAPI?format=JSON`. The 'Body' tab is selected, showing the request JSON:

```
{ 1  { 2    "degreeCode_in": "MA Single", 3    "courseList_in": [ 4      "MA522", 5      "MA622", 6      "MZ522", 7      "MC632", 8      "BSXX288YY", 9      "CSXX288YY", 10     "CSXX289YY", 11     "CSXX290YY" 12   ] 13 }
```

 The 'Response' tab shows a 200 OK status with 10 headers. The response JSON is:

```
{ 1  { 2    "__DecisionID__": "76e70ef6-aff8-492a-aa4a-15a5addf67e4", 3    "notAllowedCourses_out": [ 4      "CSXX288YY", 5      "CSXX289YY" 6    ], 7    "allowedCourses_out": [ 8      "MA522", 9      "MA622" 10   ] 11 }
```

Improving the solution





At the moment all the filtering codes are defined in the rules. So every time these change someone will need to change the rules, test and redeploy. Although that is pretty much what business rules tools were created to do, there are some circumstances where you would like to pass these values into the rules for extremely dynamic scenarios.

In this university degree example the courses will not be changing that quickly to warrant this kind of approach but imagine something like retail or financial fraud where the values could change multiple times per day. To that end the filters can be passed in as additional input parameters:

Example - Filter Courses Parameterised Operation

Input Parameters

Define the parameters required to call the execution.

Parameter name	Verbalization	Type
 degreeCode_in	DegreeCode	java.lang.String
 courseList_in	CourseList	java.lang.String[]
 courseCodeFilter1_in	CourseFilters1	java.lang.String[]
 courseCodeFilter2_in	CourseFilters2	java.lang.String[]

The rules just need a small tweak to make them use the values in the list:

```
definitions
  set 'Course' to a string in CourseList ;
if
  DegreeCode is "MA Single"
  and there is at least one string in CourseFilters1
    where Course is similar to this string ,
then
  add Course to AllowedList ;
```

```

definitions
  set 'Course' to a string in CourseList ;
if
  it is not true that DegreeCode is similar to "CS.*"
  and there is at least one string in CourseFilters2
    where Course is similar to this string ,
then
  add Course to NotAllowedList ;

```

And we have a much more flexible rule service

POST http://localhost:9090/DecisionService/rest/v1/Utilities/Filter_Courses_Parameterised/OPENAPI?format=JSON

Params	Headers	Auth	Body	Request	Response
			<pre> 1 { 2 "degreeCode_in": "MA Single", 3 "courseList_in": [4 "MA522", 5 "MA622", 6 "MZ522", 7 "MC632", 8 "BSXXX288YY", 9 "CSXXX288YY", 10 "CSXXX289YY", 11 "CSXXX290YY" 12], 13 "courseCodeFilter1_in": [14 "MA52#", "MA62#" 15], 16 "courseCodeFilter2_in": [17 "CS%288%", "CS%289%" 18] 19 } </pre>	<p>Request POST</p> <p>Response 200</p> <pre> ▶ HTTP/1.1 200 OK (10 headers) 1 { 2 "__DecisionID__": "c5ac2556-0ca9-4860 029e308830ae", 3 "notAllowedCourses_out": [4 "CSXXX288YY", 5 "CSXXX289YY" 6], 7 "allowedCourses_out": [8 "MA522", 9 "MA622" 10] 11 } </pre>	