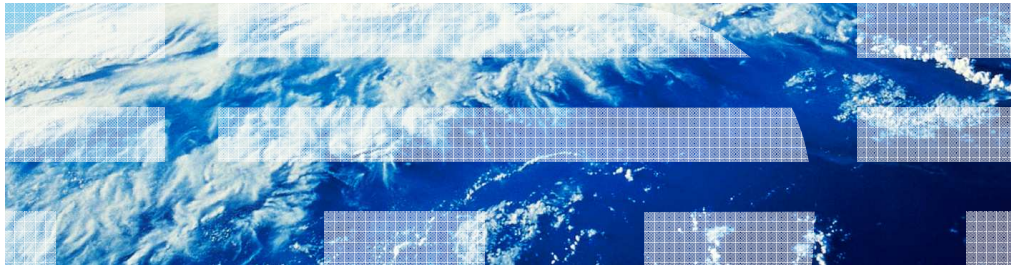


IBM Operational Decision Manager V8.0.1

Enhanced sharing of rule artifacts with COBOL



© 2012 IBM Corporation

This presentation looks at several enhancements to IBM Operational Decision Manager version 8.0.1 that enable new scenarios where rules are invoked from COBOL.

Table of contents

- Motivation for enhancements
- Generation of a Java execution object model (XOM) from a COBOL copybook
 - Preserving code added to a generate Java XOM
 - Mapping a COBOL fixed length table to a Java List
 - Mapping identical COBOL structures to same class
- Generation of a COBOL copybook from a business object model (BOM)
 - BOM with virtual attributes and virtual classes
 - Improved accessor support
 - Copybook generation with multiple BOMs

First you will learn the motivation behind these enhancements. Then you will look at the enhancements in two groups.

The first group of enhancements is related to the generation of a Java execution object model (XOM) from a COBOL copybook. One is the ability to regenerate a Java XOM from a modified copybook, without losing the code added to the original generated Java. Another is the ability to map a fixed-length COBOL table to a Java list. A third is the mapping of identical COBOL structures to the same Java class.

The second group of enhancements is related to the generation of a COBOL copybook from an existing business object model (BOM). One is support for a BOM containing virtual attributes or virtual classes. Another is better support for the handling of accessors. A third is the generation of a copybook from multiple BOMs.

Motivation

- Motivation for these enhancements:
 - Enable more scenarios allowing a rule project to be shared between COBOL clients and other types of clients.

A rule project can be deployed in such a way that there are a variety of ways to invoke it, for example, calling it as a web service, as an EJB or directly from Java. When a rule project is invoked from a COBOL program, there has to be a mapping between the COBOL copybook and the executable rules. The motivation behind the enhancements described here is to enable more scenarios where a rule project is shared between COBOL clients and other types of clients.

Preserving code added to a generated Java XOM

The first enhancement is the ability to regenerate a Java XOM from a modified copybook without losing updates that were made to the originally generated Java.

Preserving code added to a generated Java XOM (1 of 3)

- Scenario addressed
 - Initial development:
 - Start with an existing COBOL copybook
 - Generate a Java XOM from the copybook
 - Add additional imports and Java methods to the XOM
 - Generate a BOM from the Java XOM
 - At some time in the future, discover a need to modify the COBOL copybook
 - Update the COBOL copybook
 - Regenerate the Java XOM from the modified copybook
 - Resolve differences between the new Java XOM and the BOM
- Releases before V8.0.1
 - Regeneration of the Java XOM overwrites the additional imports and Java methods
 - Imports and Java methods have to be recoded before resolving differences between the Java XOM and the BOM

Here is the scenario that is addressed by this enhancement. Initially, a COBOL copybook is used to generate a Java XOM. Additional import statements and Java methods are added to the XOM and then a BOM is generated from the XOM. At a later date, the copybook needs to be updated and therefore the Java XOM must be regenerated to reflect the modifications to the copybook. Then, differences between the new Java XOM and BOM need to be resolved. This can all easily be done through Rule Designer.

In releases before version 8.0.1, when an updated copybook is used to regenerate the Java XOM, the additional import statements and Java methods are lost. As a result, you have to manually recode in the Java XOM before the Java XOM and BOM differences can be resolved. This enhancement in version 8.0.1 removes the need to do the manual recoding.

Preserving code added to a generated Java XOM (2 of 3)

copybook

```
example.cpy
01 Name.
05 firstname PIC X(10).
05 lastname PIC X(15).
```

V8.0

```

Licensed Materials - Property of IBM

package example;
import ilog.rules.bom.annotations.CustomProperties;
@CustomProperties(names = {"cobol_level", "cobol_name"},
public class Name {

    private String firstname;
    private String lastname;

    @CustomProperties(names = {"cobol_level", "cobol_na
public String getFirstname() {
    return this.firstname;
}

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    @CustomProperties(names = {"cobol_level", "cobol_na
public String getLastname() {
    return this.lastname;
}

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
}

```

V8.0.1

```

Licensed Materials - Property of IBM

package example;
import ilog.rules.bom.annotations.CustomProperties;
//@USER_IMPORT_BEGIN
//@USER_IMPORT_END
@CustomProperties(names = {"cobol_type"}, values = {"true"})
public class Name {

    private String firstname;
    private String lastname;

    public Name() {
    }

    @CustomProperties(names = {"cobol_level", "cobol_name"},
public String getFirstname() {
    return this.firstname;
}

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    @CustomProperties(names = {"cobol_level", "cobol_name"},
public String getLastname() {
    return this.lastname;
}

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

//@USER_CODE_BEGIN
//@USER_CODE_END
}

```

In the upper left corner of this slide is a screen capture of a COBOL copybook. Below it is the generated Java XOM produced by version 8.0. On the right is the generated Java XOM produced by version 8.0.1. The only difference is that, in version 8.0.1, there are comments added, which serve as markers for where to add the additional import statements and Java methods. These markers are `//@user_import_begin` and `end`, and `//@user_code_begin` and `end`.

Preserving code added to a generated Java XOM (3 of 3)

Original copybook

```
example.cpy
01 Name.
   OS firstname PIC X(10).
   OS lastname PIC X(15).
```

```

Licensed Materials - Property of IBM
package example;
import ilog.rules.bom.annotations.CustomProperties;

//CUSTOMER_IMPORT_BEGIN
import java.text.*;
//CUSTOMER_IMPORT_END

@CustomProperties(names = {"cobol_type"}, values = {"true"})
public class Name {
    private String firstname;
    private String lastname;
    public Name() {
    }
    @CustomProperties(names = {"cobol_level", "cobol_name"},
public String getFirstname() {
    return this.firstname;
}
    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }
    @CustomProperties(names = {"cobol_level", "cobol_name"},
public String getLastname() {
    return this.lastname;
}
    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
//CUSTOMER_CODE_BEGIN
    public String getFullName() {
        return firstname + lastname;
    }
//CUSTOMER_CODE_END
}

```

Updated copybook

```
example.cpy
01 Name.
   OS firstname PIC X(10).
   OS lastname PIC X(15).
   OS age PIC 9(3).
```

```

Licensed Materials - Property of IBM All Rights Reserved
package example;
import ilog.rules.bom.annotations.CustomProperties;

//USER_IMPORT_BEGIN
import java.text.*;
//USER_IMPORT_END

@CustomProperties(names = {"cobol_type"}, values = {"true"})
public class Name {
    private String firstname;
    private String lastname;
    private short age;
    public Name() {
    }
    @CustomProperties(names = {"cobol_level", "cobol_name"},
public String getFirstname() {
    return this.firstname;
}
    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }
    @CustomProperties(names = {"cobol_level", "cobol_name"},
public String getLastname() {
    return this.lastname;
}
    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
    @CustomProperties(names = {"cobol_level", "cobol_name"},
public short getAge() {
    return this.age;
}
    public void setAge(short age) {
        this.age = age;
    }
//USER_CODE_BEGIN
    public String getFullName() {
        return firstname + lastname;
    }
//USER_CODE_END
}

```

7

Enhanced sharing of rule artifacts with COBOL

© 2012 IBM Corporation

On this slide, the original copybook is in the upper left. Shown in the center of the slide is an updated version of the copybook that contains an additional field, age. On the left side is the originally generated Java XOM, which contains a manually added import statement and Java method. On the right side is the regenerated Java XOM. You can see that the additional import statement and Java method are preserved, and that new accessors for the additional age field from the copybook have been added.

Mapping a COBOL fixed length table to a Java List

The next enhancement allows a COBOL fixed length table to be mapped to a Java list.

Mapping a COBOL fixed length table to a Java List (1 of 2)

- Scenario addressed
 - A COBOL copybook that contains:
 - A fixed length table
 - A numeric count element associated with the table
 - You are generating a Java XOM from the copybook
 - You want the table to be mapped to a Java List (Java Array is the default)
- Releases before V8.0.1
 - The mapping can only be to a Java Array

In this scenario, you have a COBOL copybook that contains a fixed length table and a numeric count element associated with the table. When generating Java XOM from this copybook, you want the table to be represented as a Java list. In version 8.0, your only option was to have the table represented as a Java array. That is also the default in version 8.0.1.

Mapping a COBOL fixed length table to a Java List (2 of 2)

COBOL copybook

```
01 messages.
02 msg-count pic 9(9).
02 msgs pic x(40) value SPACE occurs 10 Times.
```

Import COBOL XOM

Enter a name for your COBOL Execution Object Model (XOM) and list the copybooks you want to include.

Execution Object Model name: messages-xom

Package Name	Source Copybook	Copy Replacing
messages	platform/messages/messages.cpy	No

Generated Java XOM

```
package messages;

import ilog.rules.bom.annotations.CustomProperties;
import java.util.List;

//@CUSTOMER_IMPORT_BEGIN
//@CUSTOMER_IMPORT_END
@CustomProperties(names = {"cobol_type"}, values = {"true"})
public class Messages {

    private List<String> msgs;

    public Messages() {
    }

    @CustomProperties(names = {"cobol_level", "cobol_name"},
    public List<String> getMsgs() {
        return this.msgs;
    }

    public void setMsgs(List<String> msgs) {
        this.msgs = msgs;
    }

//@CUSTOMER_CODE_BEGIN
//@CUSTOMER_CODE_END
}
```

COBOL Name	Reference	Java Type	Converter Applied	Rules Attribute Name
messages		messages.Messages		messages
msg-count		int	No	msg_count
msgs		java.lang.String[]	No	msgs
		java.lang.String	No	
		<Element>		

COBOL Name	Reference	Java Type	Converter Applied	Rules Attribute Name
messages		messages.Messages		messages
msg-count		int	No	msg_count
msgs		java.lang.String[]	No	msgs
		java.lang.String	No	
		java.util.List<java.lang.String>	No	
		java.lang.String	No	

COBOL Name	Reference	Java Type	Converter Applied	Rules Attribute Name
messages		messages.Messages		messages
msg-count		int	No	msg_count
msgs	messages.msg-count	java.util.List<java.lang.String>	No	msgs
		java.lang.String	No	
		<Element>		

Customized List Reference

Choose a reference for the customized list.

COBOL Name
messages
msg-count

COBOL Name	Reference	Java Type	Converter Applied	Rules Attribute Name
messages		messages.Messages		messages
msg-count		int	No	msg_count
msgs	messages.msg-count	java.util.List<java.lang.String>	No	msgs
		java.lang.String	No	
		<Element>		

10

Enhanced sharing of rule artifacts with COBOL

© 2012 IBM Corporation

This slide shows the series of screen captures for generating the Java list in the XOM.

In the upper left, you can see that the COBOL copybook contains a 10-element array, called `msgs`, and an integer count field, called `msg-count`. The dialog for generating the Java XOM from the COBOL copybook is shown in the top center of the slide. Down the left side is a series of screen captures showing the steps to do the mapping. At first, the field `msgs` is mapped to a Java String array. But using a dropdown, `msgs` can be changed to a Java List. In the next screen capture, you can see the red error indicator next to `msgs`. This is because you must have a count field when mapping to a List. From the Reference field for `msgs`, you can open the dialog and select the field to be used as the count. The result is shown in the lower left. On the right is the generated Java XOM with `msgs` mapped to a Java List.

Mapping identical COBOL structures to same class

The next enhancement allows identical COBOL structures to be mapped to the same Java class in the XOM.

Mapping identical COBOL structures to same class (1 of 2)

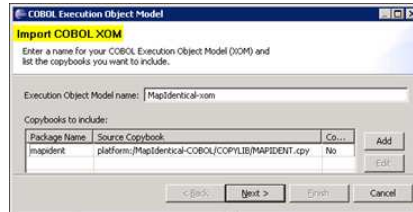
- Scenario addressed
 - A COBOL copybook that contains identical structures
 - You are generating a Java XOM from the copybook
 - You want the structures to map to the same generate class in the Java XOM
- Releases before V8.0.1
 - Each structure mapped to a unique class

This scenario applies when you have a COBOL copybook that contains identical structures and you are generating the Java XOM from the copybook. To prevent the proliferation of Java classes in the generated code, you want to map the identical structures to the same Java class.

Mapping identical COBOL structures to same class (2 of 2)

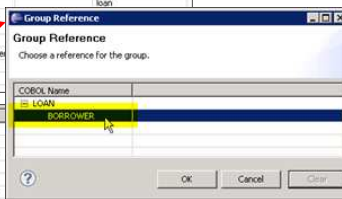
COBOL copybook

```
01 LOAN.
  05 BORROWER.
    10 NAME PIC X(20).
    10 BIRTHDAY PIC X(8).
  05 COBORROWER.
    10 NAME PIC X(20).
    10 BIRTHDAY PIC X(8).
```



COBOL Name	Reference	Java Type	Converter Applied	Rules Attribute Name
MAPIDENT				
LOAN		mapident.Loan		loan
BORROWER		mapident.Borrower	No	borrower
NAME		java.lang.String	No	name
BIRTHDAY		java.lang.String	No	birthday
COBORROWER		mapident.Coborrower	No	coborrower
NAME		java.lang.String	No	name
BIRTHDAY		java.lang.String	No	birthday

COBOL Name	Reference	Java Type	Converter Applied	Rules Attribute Name
MAPIDENT				
LOAN		mapident.Loan		loan
BORROWER		mapident.Borrower		
NAME		java.lang.String		
BIRTHDAY		java.lang.String		
COBORROWER	LOAN.BORROWER	mapident.Coborrower		
NAME		java.lang.String		
BIRTHDAY		java.lang.String		



COBOL Name	Reference	Java Type
MAPIDENT		
LOAN		mapident.Loan
BORROWER		mapident.Borrower
NAME		java.lang.String
BIRTHDAY		java.lang.String
COBORROWER	LOAN.BORROWER	mapident.Borrower
NAME		java.lang.String
BIRTHDAY		java.lang.String

Generated Java XOM

```
package mapident;
public class Loan {
    private Borrower borrower;
    private Borrower coborrower;
```

```
package mapident;
public class Borrower {
    private String name;
    private String birthday;
```

13

Enhanced sharing of rule artifacts with COBOL

© 2012 IBM Corporation

This slide shows the series of screen captures for mapping identical structures to the same Java class.

In the upper left, you see that the COBOL copybook contains two identical structures, BORROWER and COBORROWER. The dialog for generating the Java XOM from the COBOL copybook is shown in the top center of the slide. Down the left side is a series of screen captures showing the steps of the mapping. In the first panel, the structure BORROWER is mapped to the Java type Borrower, and COBORROWER is mapped to the Java type Coborrower. In the next panel, the Reference field is used to open a dialog where you can specify that COBORROWER references BORROWER. As a result, the bottom panel shows that COBORROWER is now mapped to the Java class Borrower.

On the right side, you can see that the generated Loan class contains two variables, borrower and coborrower, each of which is of type Borrower. Below that is the generated Borrower class.

BOM with virtual attributes and virtual classes

The next few enhancements are related to the scenario where there is an existing business object model (BOM) from which you generate a COBOL copybook. The first of these deals with a BOM that contains virtual attributes or virtual classes.

BOM with virtual attributes and virtual classes

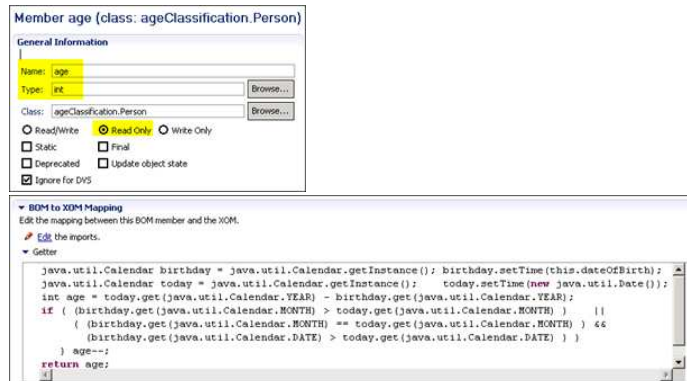
- Scenario addressed
 - Starting point – A rule project with:
 - A Java based XOM
 - A BOM containing a virtual attribute or a virtual class
 - Requirement:
 - Need to invoke the rule project from COBOL
- Releases before V8.0.1
 - The COBOL copybook and marshaller is not correctly generated

In this scenario, the Java XOM exists and is associated with a BOM containing a virtual attribute or a virtual class. You want to call this rule project from COBOL; so you need to generate a COBOL copybook.

In releases before version 8.0.1, you can't generate a COBOL copybook in this scenario because the copybook and marshaller are not correctly generated for virtual attributes and virtual classes.

BOM with virtual attribute

- What is a virtual attribute?
 - An attribute defined in the BOM that does not exist in the XOM
- Example – BOM attribute 'age' where XOM only contains 'dateOfBirth'



16

Enhanced sharing of rule artifacts with COBOL

© 2012 IBM Corporation

In order to understand this enhancement, the next few of slides provide an explanation of virtual attributes and virtual classes.

A virtual attribute is one that is defined in the business object model but does not exist in the Java execution object model. The example shown here is a BOM with an attribute, age, where the Java XOM only contains the date of birth. The top screen capture shows that, in the BOM, the attribute age is defined as a read-only integer. The bottom screen capture defines the getter method for the attribute. It contains the code that is necessary to compute someone's age, given their date of birth.

BOM with virtual class

- What is a virtual class?
 - A class defined in the BOM that does not exist in the XOM
- Possible use:
 - Define a domain of static references (enumerated values) in the BOM
 - Associate with a field in the XOM without value restrictions
- Example:
 - XOM contains field → **java.lang.String color**
 - BOM contains class → **MyColors** with static final members **'red', 'green', 'blue'**
 - BOM attribute **color** defined as type **MyColors** rather than as type **String**

Similar to a virtual attribute, a virtual class is one that exists in the business object model but does not exist in the Java execution object model.

The possible use of a virtual class is where you want to treat an attribute as a set of enumerated values in the BOM, even though its corresponding execution object model field has no value restrictions.

To illustrate this with an example, assume you have a Java XOM with a String field named color. In the BOM, you can have a virtual class such as MyColors, which has a set of static final members red, green and blue. In the BOM, the color attribute can then be defined to be of type MyColors rather than of type String. As a result, the color attribute can only contain one of the strings "red," "green," or "blue."

BOM with virtual class example definition

The screenshot displays the IBM Rational Developer for System z interface, showing the definition of a virtual class and its members in the BOM (Business Object Model).

XOM (Execution Object Model) Definition:

```

public class Person
{
    public String name;
    public Date dateOfBirth;
    private String ageGroup;

    public String getAgeGroup() {
        return ageGroup;
    }

    public void setAgeGroup(String group) {
        ageGroup = group;
    }
}
    
```

BOM (Business Object Model) Definition:

Member ageGroup (class: ageClassification Person)

- General Information:** Name: ageGroup, Type: ageClassification.Group, Class: ageClassification.Person.
- Member Verbalization:** Remove the verbalization, Create a navigation phrase, Create an action phrase, Edit the subject used in phrases.
- Navigation:** "the age group of a person" (Template: (age group) of (this))
- Action:** "set the age group of a person to a group" (Template: set the age group of (this) to (age group))

Class Group (package: ageClassification)

- General Information:** Name: Group, Namespace: ageClassification, Superclasses: java.lang.Object, Interfaces: (empty).
- Class Verbalization:** Remove the verbalization, Edit the documentation, Generate automatic variable, Term: group, Edit term.
- Members:** Adult, Baby, Child, Senior, Teen, Group().
- Domain:** Create and edit a domain for this class, Edit the domain, Restore the domain.
- Domain type: Static References:** Synchronize, Adult, Baby, Child, Senior, Teen.
- BOM to XOM Mapping:** Edit the mapping between this BOM class and the XOM. Execution name: java.lang.String, Extender name: (empty).

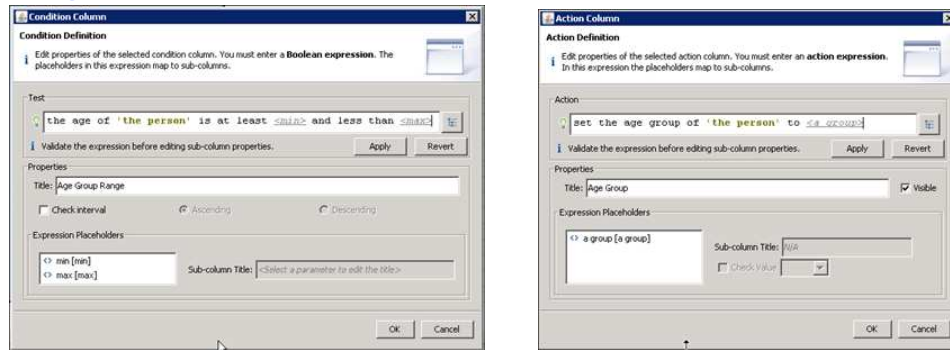
Member Child (class: ageClassification.Group)

- General Information:** Name: Child, Type: ageClassification.Group, Class: ageClassification.Group.
- Member Verbalization:** Remove the verbalization, Label: Child.
- BOM to XOM Mapping:** Edit the mapping between this BOM member and the XOM. Getter: return "Child";

18 Enhanced sharing of rule artifacts with COBOL © 2012 IBM Corporation

Here is another example of a virtual class. The execution object model shown in the upper left, has a class Person containing a String variable called ageGroup. In the upper right, you can see the definition in the BOM for the member ageGroup. Notice that the type of ageGroup is defined to be Group rather than String. On the left is the definition for the class Group, which is a virtual class since it does not exist in the execution object model. Group is composed of members Baby, Child, Teen, Adult and Senior, which are all defined as static final members. Each has a getter that returns an appropriate string. For example, the one shown on the right is for the member Child; and its getter returns the string "Child". In the lower left corner you can see that the class Group has a BOM to XOM mapping of type String. The result of using this virtual class is that the variable ageGroup, although defined as a String in the execution object model, can only contain one of the values defined in the class Group.

Using a virtual attribute and virtual class in a decision table



Virtual attribute used in condition test

Virtual class used in action

	min	≤ 2	max	Age Group
1				Baby
2	3		13	Child
3	13		20	Teen
4	20		65	Adult
5		≥ 65		Senior

19

Enhanced sharing of rule artifacts with COBOL

© 2012 IBM Corporation

On this slide you can see the virtual attribute and virtual class from the previous slides being used in a decision table. The left column of the decision table is the condition test and makes use of the age virtual attribute. The right column is the action and sets the member ageGroup, using the values defined in the virtual class Group.

The result is that, given a person's date of birth, this decision table can set the value for the person's age group.

Improved accessor support

The next enhancement provides improved support for accessors.

Improved accessor support (1 of 2)

- Scenario addressed
 - Starting point – A rule project with:
 - A Java based XOM
 - Public attributes with no explicit accessor methods
 - Boolean attribute with 'isXXXX' accessor rather than 'getXXXX'
 - Requirement:
 - Need to invoke the rule project from COBOL
- Releases before V8.0.1
 - The COBOL marshaller is not correctly generated

The scenario addressed by this enhancement is a rule project that has an existing Java execution object model. The Java XOM has public attributes but does not have getter and setter methods explicitly defined for those attributes. In addition, there is a Boolean attribute that has an “is” accessor method rather than a “get” accessor method. In releases before version 8.0.1, this Java execution object model can't be used when calling the rule project from COBOL because the marshalling code is not correctly generated.

Improved accessor support (2 of 2)

V8.0

```
public class Member {  
  
    private String name;  
    private int age;  
    private boolean married;  
  
    public String getName() {  
        return this.name;  
    }  
    public void setName(String n) {  
        this.name = n;  
    }  
  
    public int getAge() {  
        return this.age;  
    }  
    public void setAge(int a) {  
        this.age = a;  
    }  
  
    public boolean getMarried() {  
        return this.married;  
    }  
    public void setMarried(boolean m) {  
        this.married = m;  
    }  
}
```

V8.0.1

```
public class Member {  
  
    public String name;  
    public int age;  
    private boolean married;  
  
    public boolean isMarried() {  
        return this.married;  
    }  
    public void setMarried(boolean m) {  
        this.married = m;  
    }  
}
```

This slide shows an example of a Java XOM that has attributes for name (a String), age (an int) and married (a Boolean). On the left is the code required in version 8.0 to make the XOM compatible for use with COBOL. Each of the attributes has a private variable and explicitly declared getter and setter methods. In version 8.0.1, the XOM on the left still works; but alternatively, the XOM on the right also works. In this case, the name and age attributes are public variables with no explicit getter and setter methods. And the getter method for the Boolean variable married is defined as isMarried rather than getMarried.

Copybook generation with multiple BOMs

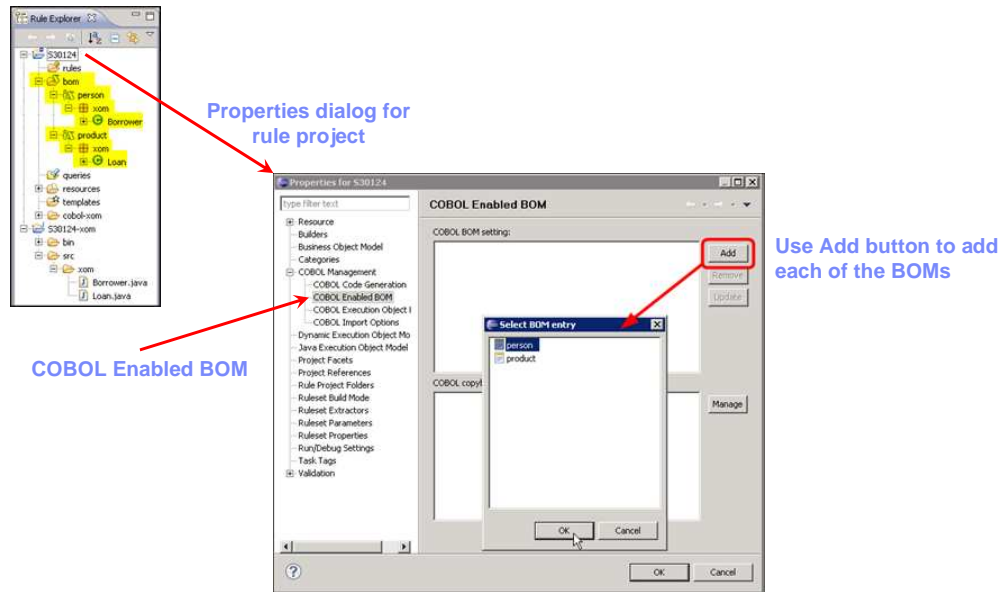
This enhancement affects the ability to generate a COBOL copybook from multiple business object models.

Copybook generation with multiple BOMs (1 of 5)

- Scenario addressed
 - Starting point:
 - A rule project with more than one BOM
 - Requirement:
 - Use a single COBOL copybook when invoking the rule project
- Releases before V8.0.1
 - A COBOL copybook can only be generated from a single BOM

The starting point for this scenario is a rule project that has more than one BOM and the requirement that only one copybook be used when calling these rules from COBOL. In releases before version 8.0.1, separate copybooks have to be generated, one for each BOM. The next few slides show you how to generate a single copybook from multiple BOMs.

Copybook generation with multiple BOMs (2 of 5)



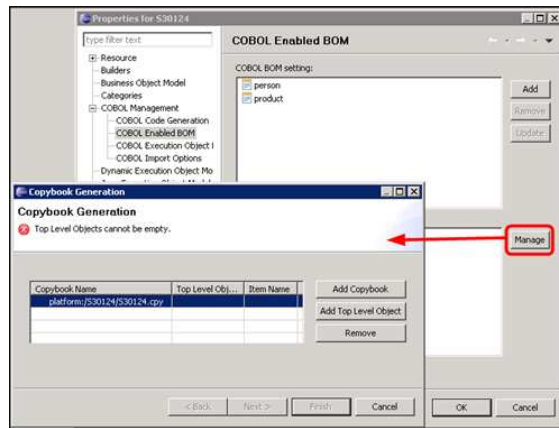
25

Enhanced sharing of rule artifacts with COBOL

© 2012 IBM Corporation

In the upper left is a rule project with two BOMs. One BOM is called person and has a class Borrower; and the other BOM is called product and has a class Loan. The Properties dialog for the rule project is shown in the middle of the slide. To begin the copybook generation, navigate to COBOL Management, and COBOL Enabled BOM. Use the Add button to add a BOM. Do this for each of the BOMs.

Copybook generation with multiple BOMs (3 of 5)



The Manage button will open the Copybook Generation dialog

Top Level Objects still need to be selected

After both BOMs have been added, use the Manage button to open the Copybook Generation dialog. Initially the dialog reports an error because no Top Level Objects have been specified yet.

Copybook generation with multiple BOMs (4 of 5)

Types

Choose a type (? = any character, * = any character)

Matching types:

- Borrower
- Loan

Qualifiers:

zom - (/S30124/bom/person.bom)

Copybook Name | **Top Level Object** | **Item Name** | **Add Copybook**

Copybook Name	Top Level Object	Item Name	Add Copybook
platform/S30124/S30124.cpy			Add Top Level Object Remove

Clicking Add Top Level Object will open Types dialog

Select the right type

Types

Choose a type (? = any character, * = any character)

Matching types:

- Borrower
- Loan

Qualifiers:

zom - (/S30124/bom/product.bom)

Copybook Name | **Top Level Object** | **Item Name** | **Add Copybook**

Copybook Name	Top Level Object	Item Name	Add Copybook
platform/S30124/S30124.cpy	zom.Borrower	borrower	Add Top Level Object Remove

Clicking Add Top Level Object will open Types dialog

Select the right type

Types

Choose a type (? = any character, * = any character)

Matching types:

- Borrower
- Loan

Qualifiers:

zom - (/S30124/bom/person.bom)

Array
 Array Type
Dimensions: 1

Copybook Name | **Top Level Object** | **Item Name** | **Add Copybook**

Copybook Name	Top Level Object	Item Name	Add Copybook
platform/S30124/S30124.cpy	zom.Borrower	borrower	Add Top Level Object Remove
	zom.Loan	loan	Add Top Level Object Remove

Now everything is correct

27

Enhanced sharing of rule artifacts with COBOL

© 2012 IBM Corporation

Use the Add Top Level Object button to add the classes from each of the BOMs. In the bottom right screen capture, you can see that the Borrower and Loan types have been added.

Copybook generation with multiple BOMs (5 of 5)

Resulting copybook maps both BOMs

```

Sharing2Minloan.py - Notepad
File Edit Format View Help
* This COPYBOOK is generated from Following BOM Entry:
* bomname
* It is created from IBM Decision Server at 2012/09/16 16:18:10.
* xom.Borrower borrower : BORROWER
* int age : AGE
* long creditscore : CREDITSCORE
* java.lang.String name : NAME
* long yearlyincome : YEARLYINCOME
* xom.Loan loan : LOAN
* long amount : AMOUNT
* boolean approved : APPROVED
* java.util.Date effectdate : EFFECTDATE
* java.util.List messages : MESSAGES
* short yearlyinterestrate : YEARLYINTERESTRATE
* long yearlyrepayment : YEARLYREPAYMENT
01 borrower.
  02 age pic 9(10).
  02 creditscore pic 9(18).
  02 name pic X(20) value SPACE.
  02 yearlyincome pic 9(18).
01 loan.
  02 amount pic 9(18).
  02 approved pic X.
  88 BoolValue value 'T'.
  02 effectdate pic 9(8).
  02 messages-Num pic 9(9).
  02 messages pic X(20) value SPACE occurs 10 Times.
  02 yearlyinterestrate pic 9(5).
  02 yearlyrepayment pic 9(18).
  
```

The resulting copybook contains the data structures for both BOMs, borrower and loan.

Summary

- Explained the motivation for these enhancements
- Looked at the enhancements for
 - Generation of a Java XOM from a COBOL copybook
 - Generation of a COBOL copybook from a BOM

In this presentation, you learned some of the motivation behind these enhancements to Operational Decision Manager. You looked at the enhancements related to generating a Java execution object model from a COBOL copybook and the enhancements related to generating a COBOL copybook from a business object model.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_ODM801_ShareArtifactsWithCOBOL.ppt

This module is also available in PDF format at: [../ODM801_ShareArtifactsWithCOBOL.pdf](http://ODM801_ShareArtifactsWithCOBOL.pdf)

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, disclaimer, and copyright information

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at <http://www.ibm.com/legal/copytrade.shtml>

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.

© Copyright International Business Machines Corporation 2012. All rights reserved.