IBM Operational Decision Manager V8.0.1

Decision engine for zRule Execution Server

This presentation introduces the new decision engine for running rules in a zRule Execution Server.

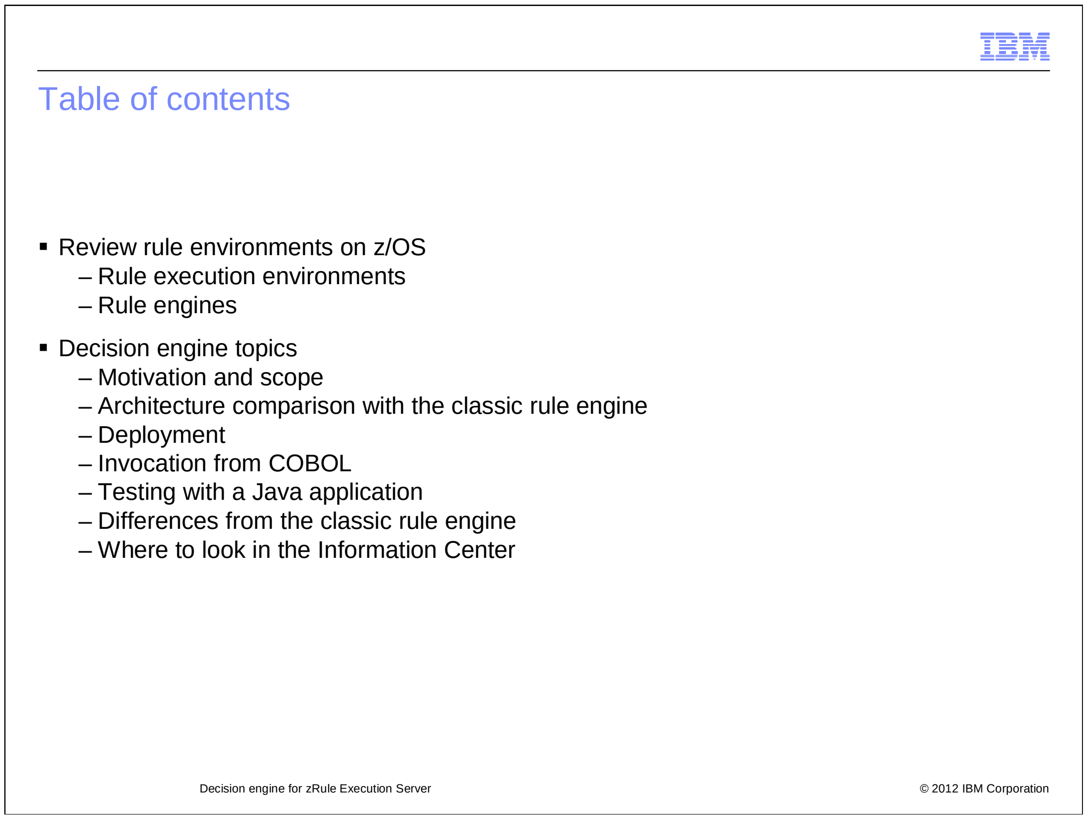
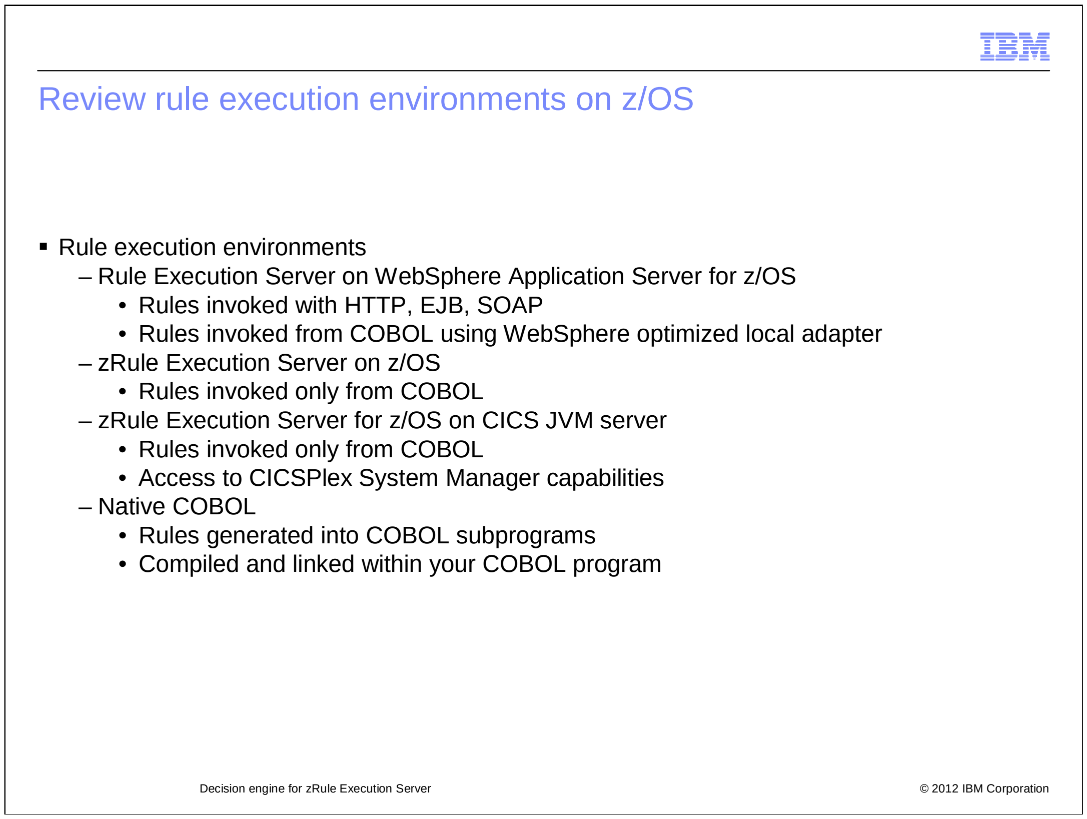


Table of contents

Decision engine for zRule Execution Server

© 2012 IBM Corporation

Before discussing the decision engine, the environments for processing rules in z/OS are reviewed. Then the motivation and applicable scope for the decision engine is described. The architecture is then compared with that of the classic rule engine. How rules using the decision engine are deployed, invoked from COBOL and tested with a Java application are presented. Some of the differences in the decision engine and classic rule engine are highlighted and a map is provided to help you find decision engine relevant material in the Information Center.



Review rule execution environments on z/OS

- Rule execution environments
  - Rule Execution Server on WebSphere Application Server for z/OS
    - Rules invoked with HTTP, EJB, SOAP
    - Rules invoked from COBOL using WebSphere optimized local adapter
  - zRule Execution Server on z/OS
    - Rules invoked only from COBOL
  - zRule Execution Server for z/OS on CICS JVM server
    - Rules invoked only from COBOL
    - Access to CICSPlex System Manager capabilities
  - Native COBOL
    - Rules generated into COBOL subprograms
    - Compiled and linked within your COBOL program

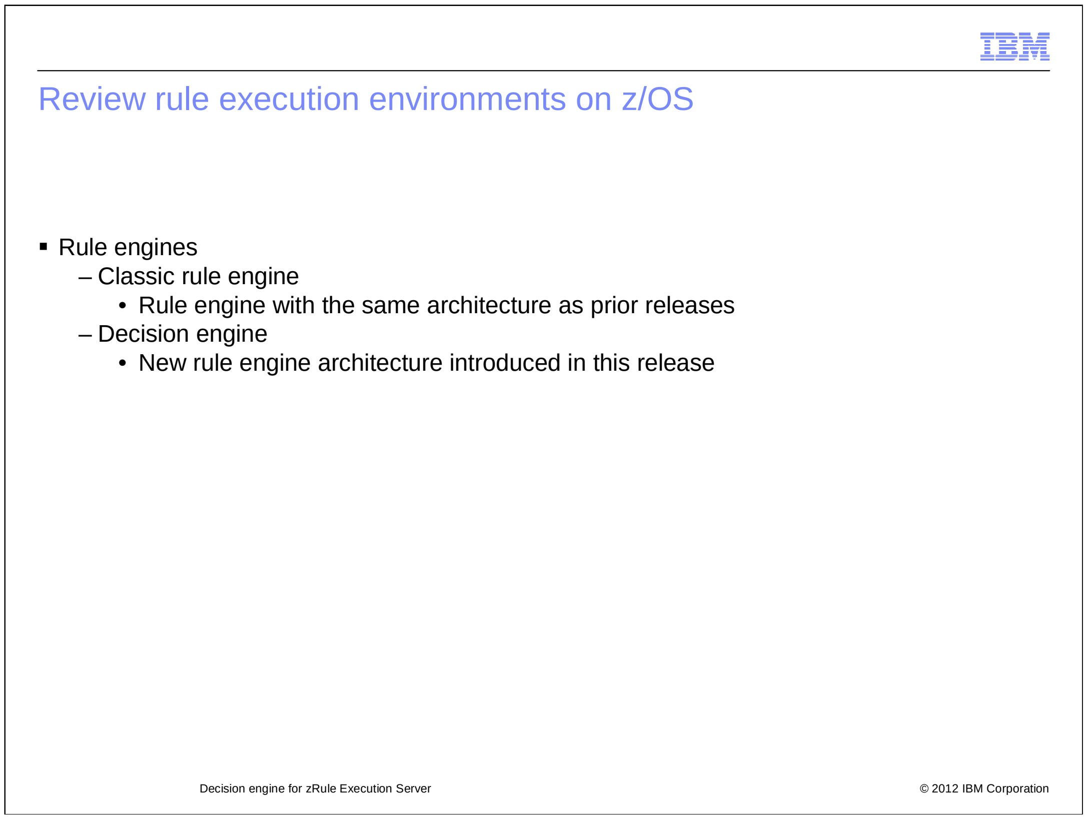
Decision engine for zRule Execution Server

There are several different environments for running rules on z/OS. The first is using the Rule Execution Server running on WebSphere Application Server. This environment is a Decision Server, and rules can be invoked in many different ways, such as using HTTP, SOAP and EJB. Rules in this environment can also be called from COBOL using the WebSphere optimized local adapter.

The next environment is the zRule Execution Server. It is used only for calling rules from COBOL programs.

The third environment is the zRule Execution Server for z/OS on CICS JVM server. This also is only used for calling rules from COBOL. However, because it is running in CICS, capabilities such as the CICSPlex System Manager are also available.

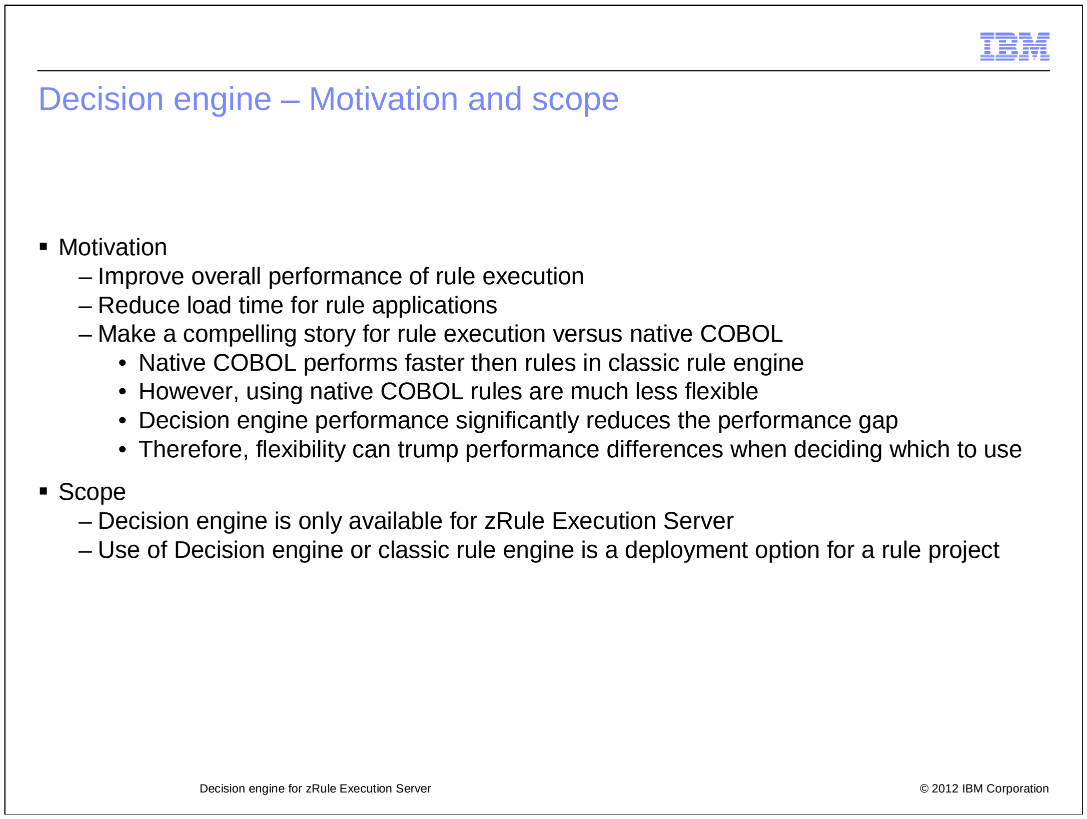
Finally, a non-server based mechanism is using native COBOL. Rules are used to generate COBOL subprograms, which are then compiled and linked with the calling COBOL programs.



Review rule execution environments on z/OS

- Rule engines
  - Classic rule engine
    - Rule engine with the same architecture as prior releases
  - Decision engine
    - New rule engine architecture introduced in this release

In addition to the environments for running rules, there are also two different runtime engines for processing rules. The first is called the classic rule engine and is the same basic implementation and architecture used for running rules in previous releases. The next is called the decision engine and is the topic of this presentation. It is a new implementation in version 8.0.1.


Decision engine – Motivation and scope

- Motivation
  - Improve overall performance of rule execution
  - Reduce load time for rule applications
  - Make a compelling story for rule execution versus native COBOL
    - Native COBOL performs faster then rules in classic rule engine
    - However, using native COBOL rules are much less flexible
    - Decision engine performance significantly reduces the performance gap
    - Therefore, flexibility can trump performance differences when deciding which to use
- Scope
  - Decision engine is only available for zRule Execution Server
  - Use of Decision engine or classic rule engine is a deployment option for a rule project

Decision engine for zRule Execution Server © 2012 IBM Corporation

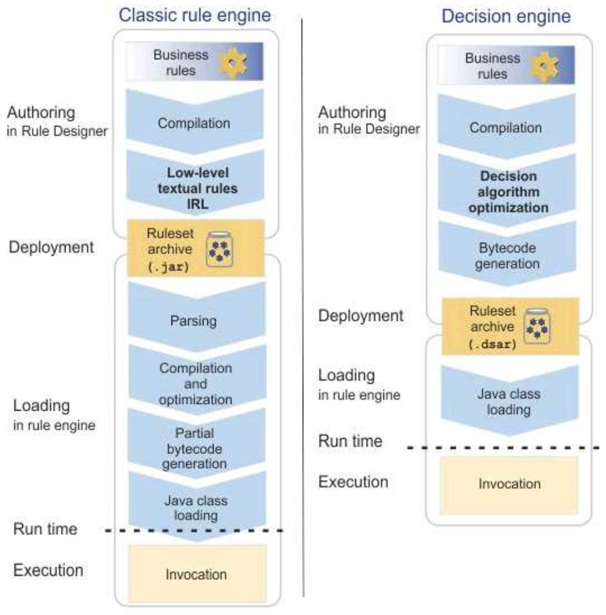
Before delving into the decision engine, this slide addresses the motivation for introducing the decision engine and defines the scope of where it applies.

The major reason for introducing the new decision engine is to provide an overall improvement in the performance of rule execution and to decrease the time needed to load a rule application.

While performance gains are good in and of themselves, there is an additional factor at play here, to make rule execution performance competitive with rules in native COBOL. Execution in native COBOL is much faster than when using the classic rule engine. However, there is much less flexibility and more overhead in maintaining rules in native COBOL when compared to deploying rules to a server environment. The decision engine significantly reduces the performance gap between server based rule execution and native COBOL. Therefore, it is no longer a significant performance tradeoff to choose the more flexible option.

The decision engine is only available in one environment, the zRule Execution Server. Also, projects deployed to the zRule Execution Server can use either the classic rule engine or the decision engine. The choice is an authoring time decision.

Decision engine architecture comparison to classic rule engine



- Different deployment archive
  - Engine selected at authoring time
- Optimization and compilation
  - Decision engine - done at authoring time
  - Classic rule engine – done at load time
- Execution
  - Decision engine – fully compiled Java
  - Classic rule engine – compiled Java and interpreted code

Decision engine for zRule Execution Server © 2012 IBM Corporation

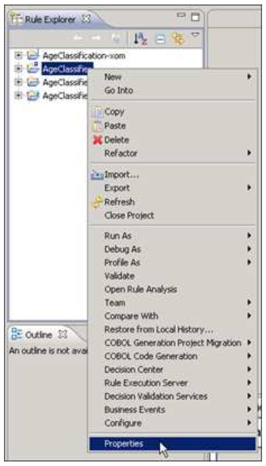
The architecture of the classic rule engine is shown on the left as compared to that on the decision engine on the right. Each has a different deployment archive and therefore the selection of which engine is used is made at authoring time.

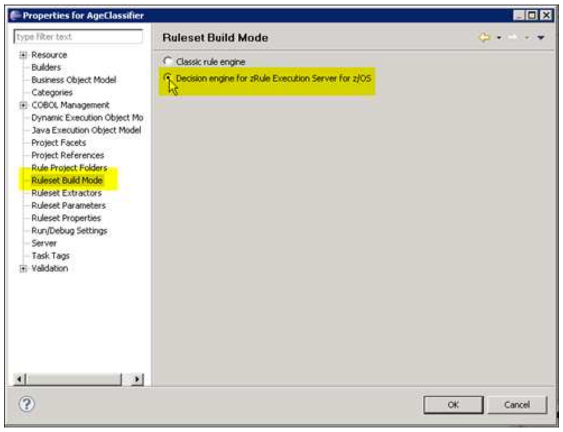
There are basically two fundamental differences between the two architectures. With the classic rule engine, the optimization and compilation of the rules is done after deployment when the rule project is loaded into the server. For the decision engine, the optimization and compilation of the rules is done at authoring time before deployment.

The other fundamental difference between the architectures affects the actual execution of the rules. With the classic rule engine, the execution of the rules is a combination of running compiled code and interpreted code. With the decision engine, the execution is fully compile code.

It is these differences in architecture that help contribute to the reduced load time and increased performance of the decision engine.

Deployment – Setting the build mode





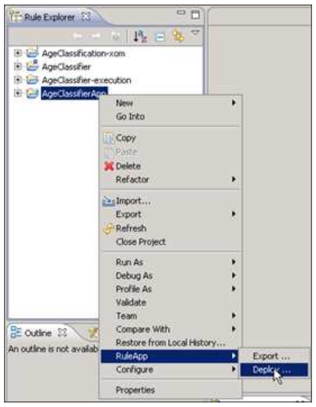
Ruleset Build Mode

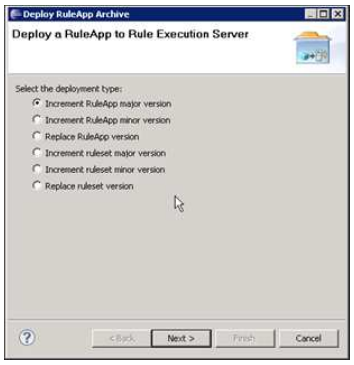
Classic rule engine [✓ Decision engine for zRule Execution Server for z/OS]

- Setting the build mode
  - – Open the **Properties** view of your rule project
  - – **Ruleset Build Mode → Decision engine for zRule Execution Server for z/OS**

The build mode determines which engine is used for your project’s rule execution. This is done by opening the Properties view of the rule project, selecting Ruleset Build Mode and then selecting which engine to use.

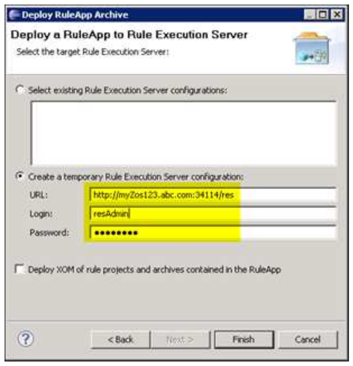
Deployment to zRule Execution Server




Deploy RuleApp Archive

Deploy a RuleApp to Rule Execution Server

Select the deployment type:
- [✓ Increment RuleApp major version]
- Increment RuleApp minor version
- Replace RuleApp version
- Increment ruleset major version
- Increment ruleset minor version
- Replace ruleset version


Deploy RuleApp Archive

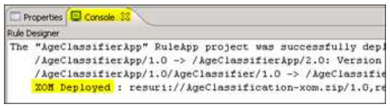
Deploy a RuleApp to Rule Execution Server

Select the target Rule Execution Server:

- Select existing Rule Execution Server configurations:
- [✓ Create a temporary Rule Execution Server configuration:]

URL: http://myZos123.abc.com:34114/res
Login: resAdmin
Password: ●●●●●●●●●

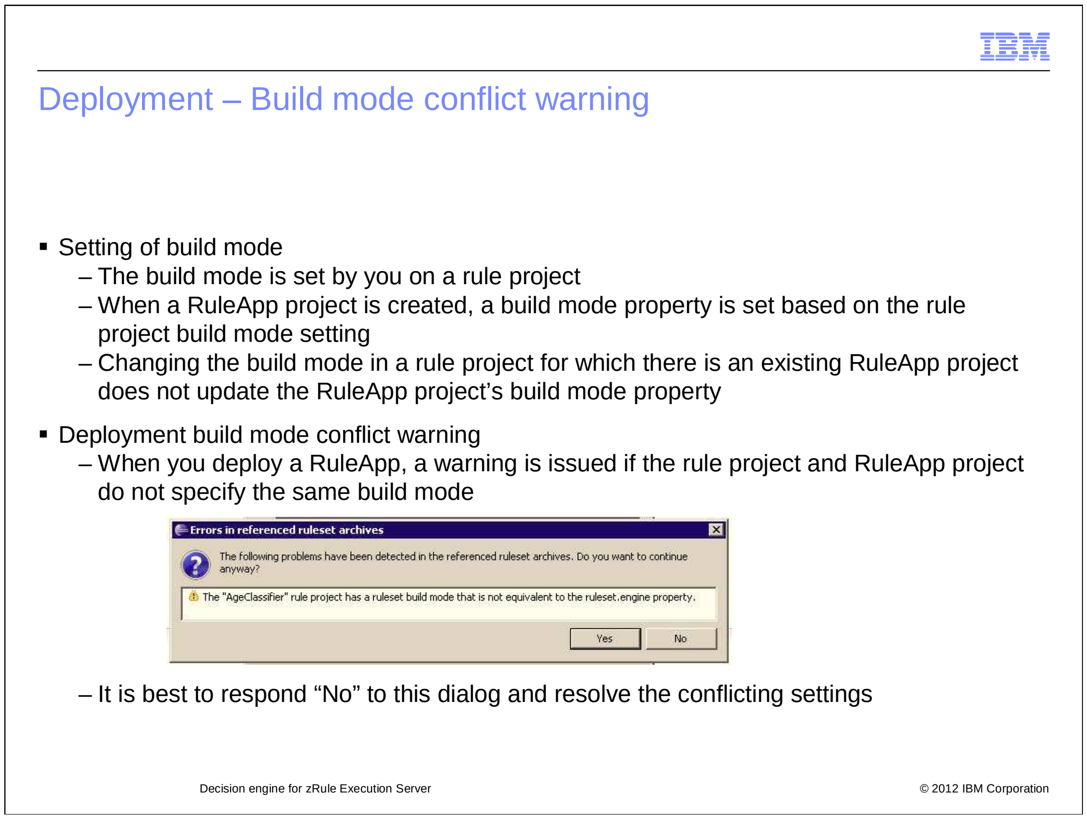
Deploy XOM of rule projects and archives contained in the RuleApp



```
Rule Designer
The "AgeClassifierApp" RuleApp project was successfully dep
    /AgeClassifierApp/1.0 -> /AgeClassifierApp/2.0: Version
    /AgeClassifierApp/1.0/AgeClassifier/1.0 -> /AgeClassifie
   XOM Deployed : resuri://AgeClassification-xom.zip/1.0,re
```

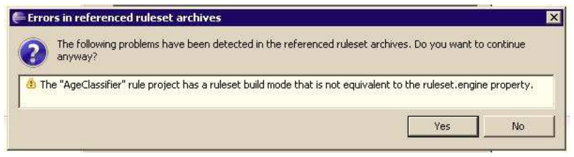
- Deploying from Rule Designer
  - – Select Deploy… from the RuleApp menu
  - – Select version handling option
  - – Provide URL and credentials for zRule Execution Server
  - – Results shown in Console view

This slide provides an example of how to deploy from Rule Designer to zRule Execution Server. You must have a rule application associated with you rule project. From the rule application’s pop-up menu, select RuleApp, then select Deploy, and the Deploy RuleApp Archive dialog opens. On the first panel select how you’d like version handling done in the deployment and on the next panel provide the URL and credentials for your zRule Execution Server. Status of the deployment is returned in the Console view of Rule Designer.



Deployment – Build mode conflict warning

- Setting of build mode
  - The build mode is set by you on a rule project
  - When a RuleApp project is created, a build mode property is set based on the rule project build mode setting
  - Changing the build mode in a rule project for which there is an existing RuleApp project does not update the RuleApp project’s build mode property
- Deployment build mode conflict warning
  - When you deploy a RuleApp, a warning is issued if the rule project and RuleApp project do not specify the same build mode



Errors in referenced ruleset archives

The following problems have been detected in the referenced ruleset archives. Do you want to continue anyway?

The "AgeClassifier" rule project has a ruleset build mode that is not equivalent to the ruleset.engine property.

  - It is best to respond “No” to this dialog and resolve the conflicting settings

Decision engine for zRule Execution Server © 2012 IBM Corporation

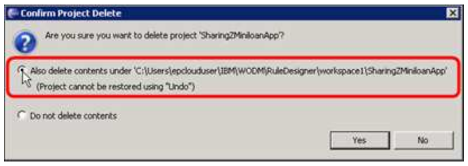
This slide provides some background about a build mode conflict warning you might encounter when deploying.

The build mode is set by you on a rule project as was described on a previous slide. When you create a rule application project, the value of the build mode property in the rule project is used to set a property in the rule application project. Subsequently, changing the build mode on the rule project does not update the property in the rule application project.

When this happens, deploying the rule application leads to an error dialog with a warning that the build mode properties of the rule project and rule application project are not equivalent. You are given the option to continue, but it best to respond no to this dialog so that you can first resolve the conflicting settings.

Deployment – Build mode conflict warning

- Resolving the conflicting settings
  – After changing the Rule project, delete and re-create the RuleApp project
  – Make sure in the delete dialog you select to delete from the file system contents



Confirm Project Delete

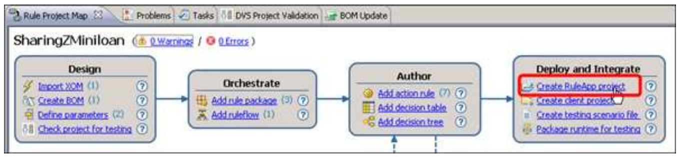
Are you sure you want to delete project 'SharingZMiniloanApp'?

Also delete contents under 'C:\Users\epclouduser\IBM\WODM\RuleDesigner\workspace1\SharingZMiniloanApp'
(Project cannot be restored using "Undo")

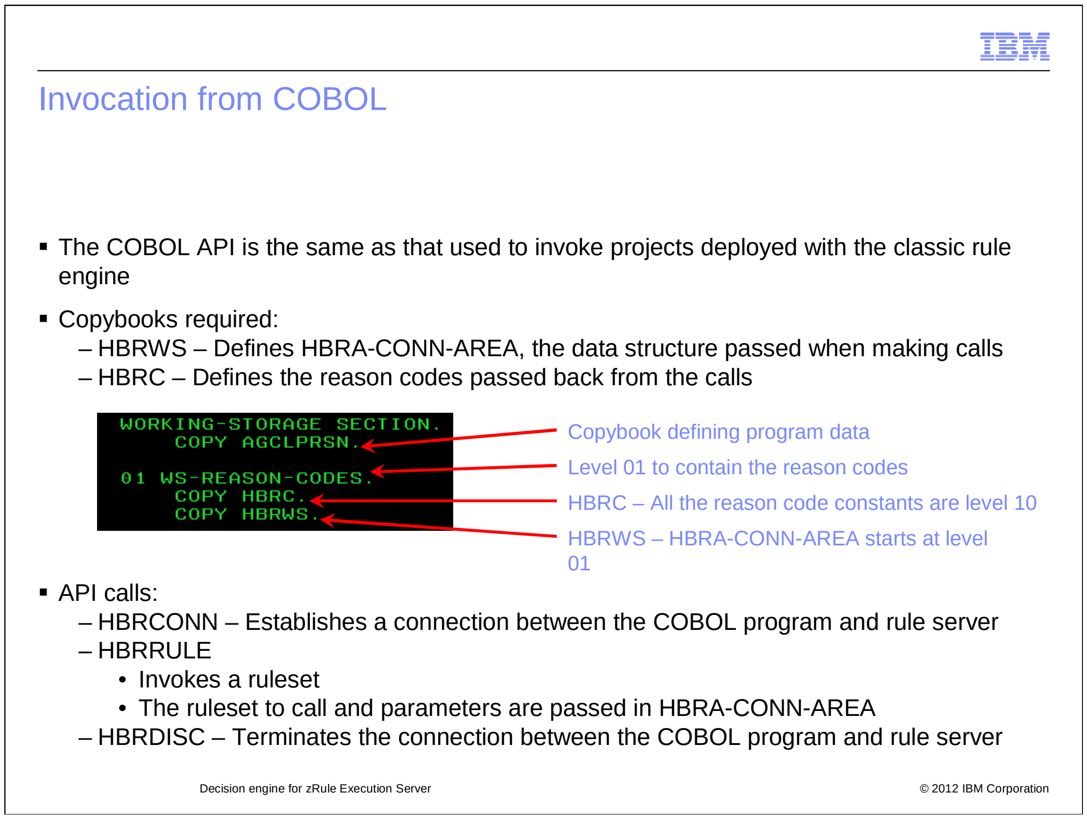
Do not delete contents

Yes No

  – Use the Rule Project Map to re-create your RuleApp



There is no mechanism provided to change the build mode property for the rule application, so the best approach is to delete and then re-create it. When deleting the rule application, make sure you select to delete the contents on the file system. Then you can use the Rule Project Map to re-create your RuleApp project.



Invocation from COBOL

- The COBOL API is the same as that used to invoke projects deployed with the classic rule engine
- Copybooks required:
  - HBRWS – Defines HBRA-CONN-AREA, the data structure passed when making calls
  - HBRC – Defines the reason codes passed back from the calls

```
  WORKING-STORAGE SECTION.
      COPY AGCLPRSN.

  01 WS-REASON-CODES.
      COPY HBRC.
      COPY HBRWS.
```

Copybook defining program data

Level 01 to contain the reason codes

HBRC – All the reason code constants are level 10

HBRWS – HBRA-CONN-AREA starts at level 01

- API calls:
  - HBRCONN – Establishes a connection between the COBOL program and rule server
  - HBRRULE
    - Invokes a ruleset
    - The ruleset to call and parameters are passed in HBRA-CONN-AREA
  - HBRDISC – Terminates the connection between the COBOL program and rule server

Decision engine for zRule Execution Server

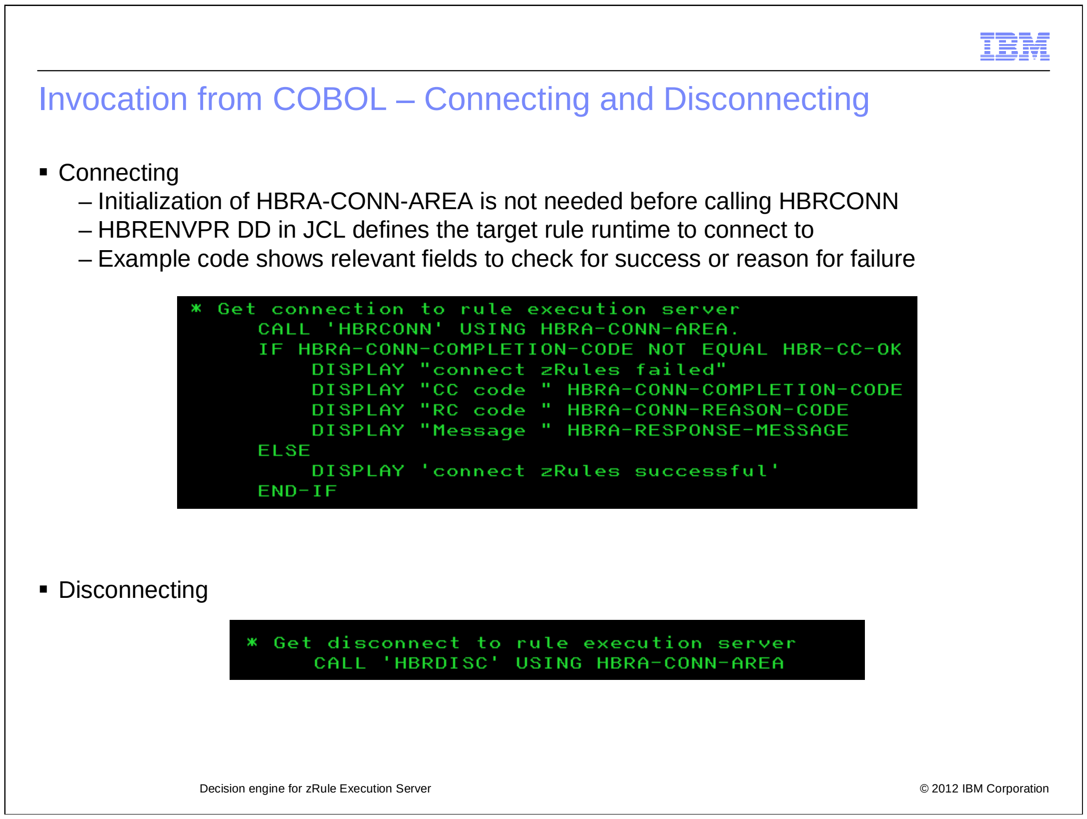
© 2012 IBM Corporation

Calling a rule project that is deployed for the decision engine uses the same COBOL APIs as rule projects deployed for the classic rule engine.

There are two required copybooks that must be included in your working storage section. The first is HBRWS which defines the HBRA-CONN-AREA data structure that is passed on the API calls. The second is HBRC which provides a set of reason code constants used to communicate status of a call.

HBRC constants are all defined at level 10, so the copybook should be pulled in within a higher level structure. HBRWS starts at level 1, so it is self contained.

The API has three calls. HBRCONN establishes the connection between the COBOL program and the rule execution server. HBRRULE invokes a ruleset with HBR_CONN_AREA used to pass the ruleset name and ruleset parameters. Finally, HBRDISC is used to terminate the connection between the COBOL program and the rule execution server.



Invocation from COBOL – Connecting and Disconnecting

- Connecting
  - Initialization of HBRA-CONN-AREA is not needed before calling HBRCONN
  - HBRENVPR DD in JCL defines the target rule runtime to connect to
  - Example code shows relevant fields to check for success or reason for failure

```
* Get connection to rule execution server
     CALL 'HBRCONN' USING HBRA-CONN-AREA.
     IF HBRA-CONN-COMPLETION-CODE NOT EQUAL HBR-CC-OK
         DISPLAY "connect zRules failed"
         DISPLAY "CC code " HBRA-CONN-COMPLETION-CODE
         DISPLAY "RC code " HBRA-CONN-REASON-CODE
         DISPLAY "Message " HBRA-RESPONSE-MESSAGE
     ELSE
         DISPLAY 'connect zRules successful'
     END-IF
```

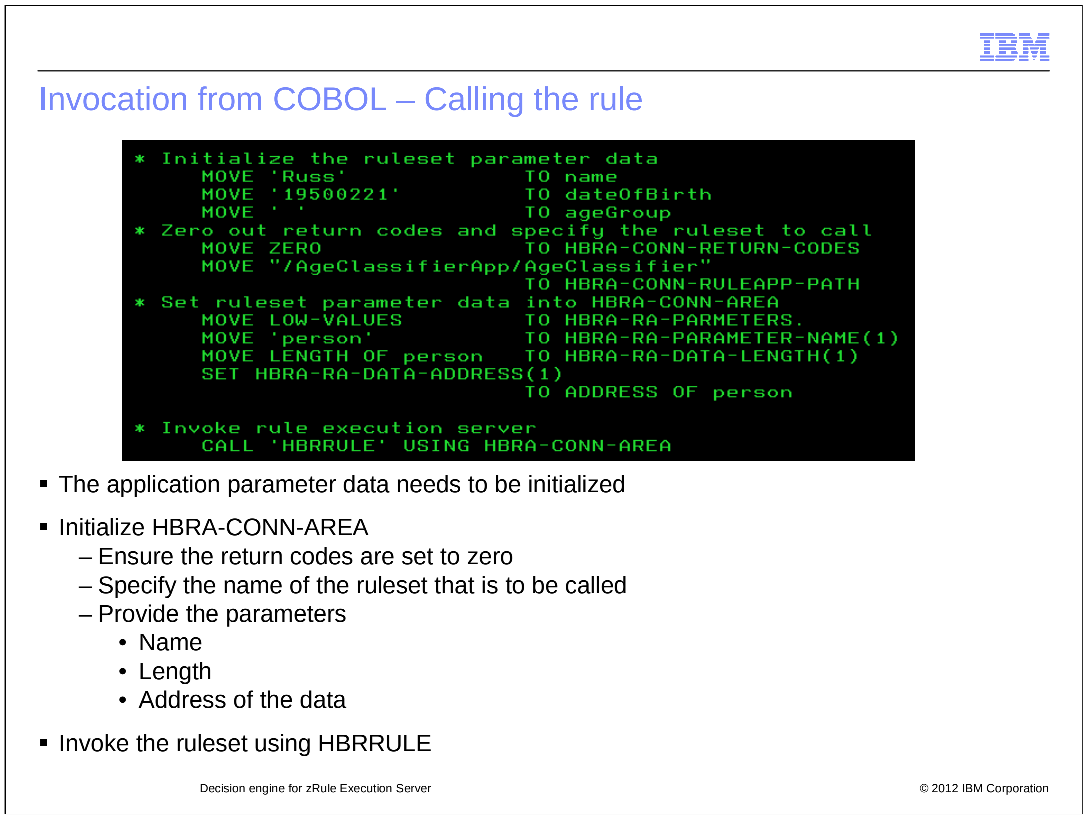
- Disconnecting

```
* Get disconnect to rule execution server
      CALL 'HBRDISC' USING HBRA-CONN-AREA
```

Decision engine for zRule Execution Server © 2012 IBM Corporation

To connect using HBRCONN, the HBRA-CONN-AREA does not need to be initialized. The rule execution server to connect to is specified by the HBRENVPR DD statement in your JCL. The example code on the slide shows how to check if the connection was successful, and which fields contain relevant information if there was a failure.

To disconnect from the rule execution server, call HBRDISC as is shown in the code sample.

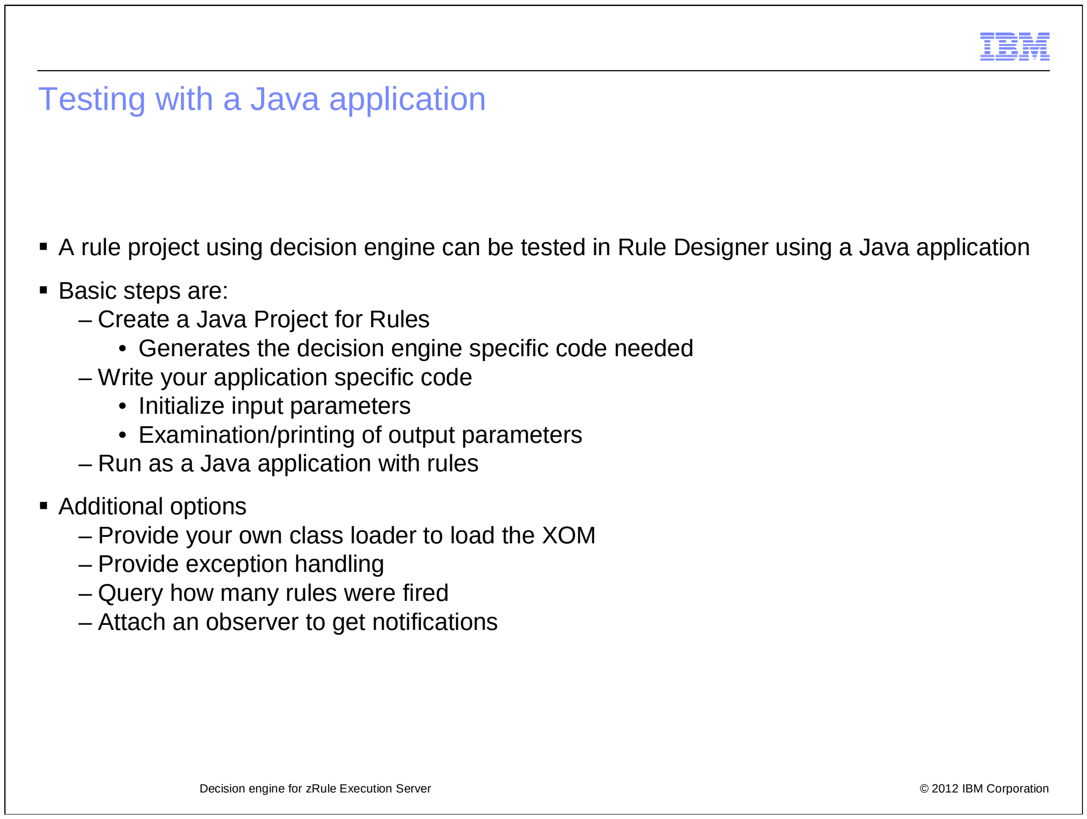


Invocation from COBOL – Calling the rule

```
* Initialize the ruleset parameter data
     MOVE 'Russ'               TO name
     MOVE '19500221'           TO dateOfBirth
     MOVE ' '                  TO ageGroup
* Zero out return codes and specify the ruleset to call
     MOVE ZERO                 TO HBRA-CONN-RETURN-CODES
     MOVE "/AgeClassifierApp/AgeClassifier"
                               TO HBRA-CONN-RULEAPP-PATH
* Set ruleset parameter data into HBRA-CONN-AREA
     MOVE LOW-VALUES           TO HBRA-RA-PARMETERS.
     MOVE 'person'             TO HBRA-RA-PARAMETER-NAME(1)
     MOVE LENGTH OF person     TO HBRA-RA-DATA-LENGTH(1)
     SET HBRA-RA-DATA-ADDRESS(1)
                               TO ADDRESS OF person

* Invoke rule execution server
     CALL 'HBRRULE' USING HBRA-CONN-AREA
```

- The application parameter data needs to be initialized
- Initialize HBRA-CONN-AREA
  - Ensure the return codes are set to zero
  - Specify the name of the ruleset that is to be called
  - Provide the parameters
    - Name
    - Length
    - Address of the data
- Invoke the ruleset using HBRRULE

To call the rule using HBRRULE there are a few things you need to do. First, the ruleset parameter need to be initialized and next the HBRA-CONN-AREA needs to be set up. This involves ensuring that the return codes are set to zero, that the name of the ruleset to call is provided, and that the name, length and address of each parameter is set. Once this is done, you then call HBRRULE. Following the invocation, you need to check the return codes. The code to do this is the same as the code for HBRCONN shown on the previous slide.



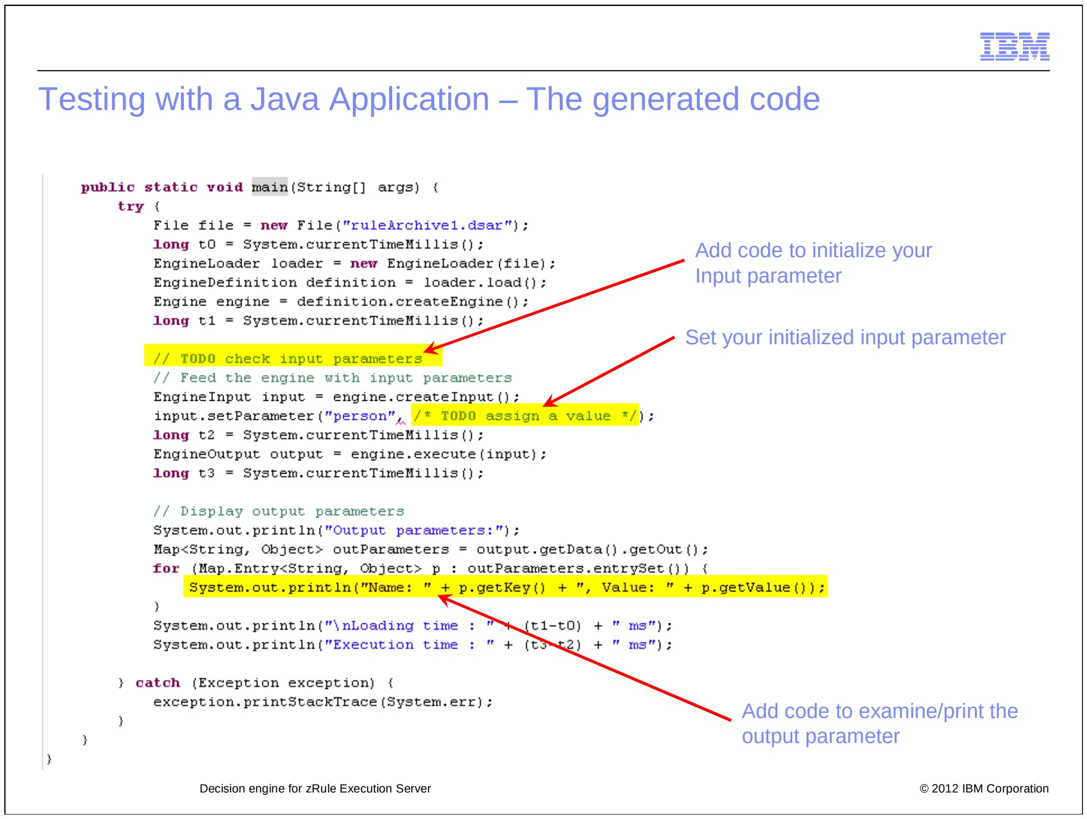
Testing with a Java application

- A rule project using decision engine can be tested in Rule Designer using a Java application
- Basic steps are:
  - Create a Java Project for Rules
    - Generates the decision engine specific code needed
  - Write your application specific code
    - Initialize input parameters
    - Examination/printing of output parameters
  - Run as a Java application with rules
- Additional options
  - Provide your own class loader to load the XOM
  - Provide exception handling
  - Query how many rules were fired
  - Attach an observer to get notifications

Decision engine for zRule Execution Server

A rule project that uses the decision engine can be tested in Rule Designer using a Java application. First you create a Java Project for Rules which generates the decision engine specific code needed to invoke the rules. Within this generated code, you then add your application specific code that initializes the input parameters and examines or prints the output parameters. The Java code is then run in Rule Designer by selecting run as a Java application with rules.

In addition, there are other things you can do when testing this way. There is the ability to provide your own class loader for the execution object model. Also, you can include exception handling, query how many rules were fired, and attach an observer to get notifications.



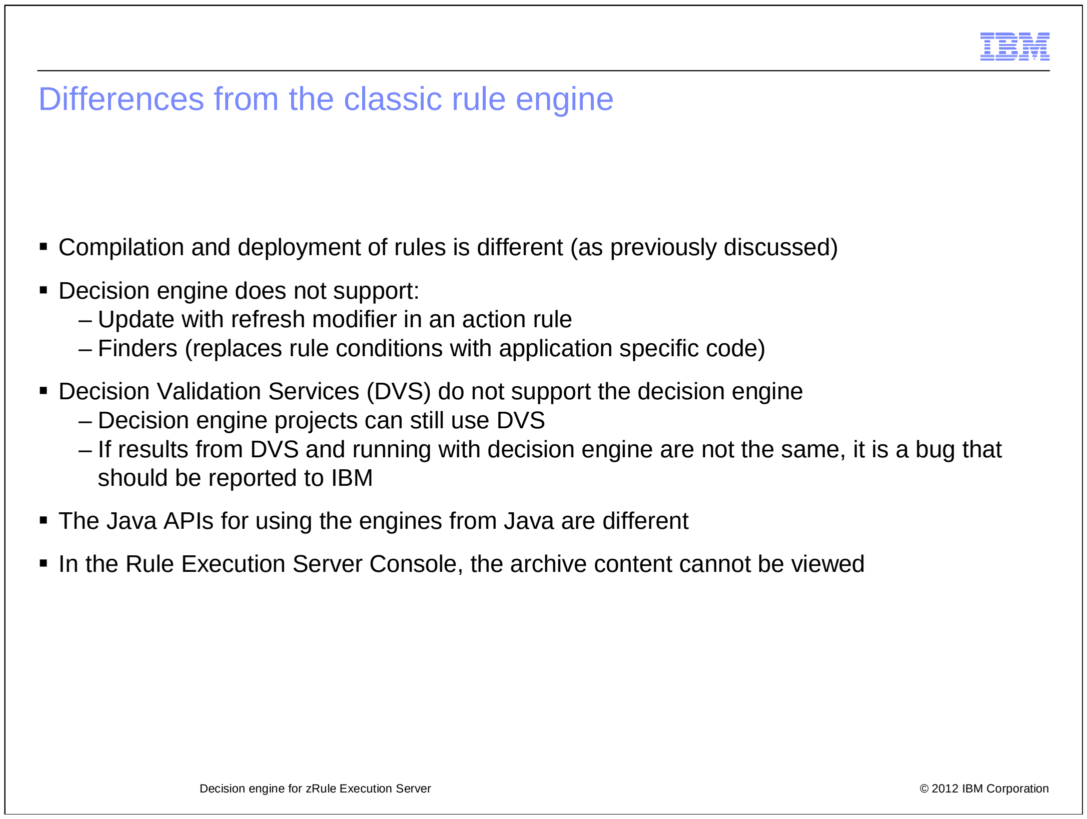
Testing with a Java Application – The generated code

```
public static void main(String[] args) {
    try {
        File file = new File("ruleArchive1.dsar");
        long t0 = System.currentTimeMillis();
        EngineLoader loader = new EngineLoader(file);
        EngineDefinition definition = loader.load();
        Engine engine = definition.createEngine();
        long t1 = System.currentTimeMillis();

        // TODO check input parameters
        // Feed the engine with input parameters
        EngineInput input = engine.createInput();
        input.setParameter("person", /* TODO assign a value */);
        long t2 = System.currentTimeMillis();
        EngineOutput output = engine.execute(input);
        long t3 = System.currentTimeMillis();

        // Display output parameters
        System.out.println("Output parameters:");
        Map<String, Object> outParameters = output.getData().getOut();
        for (Map.Entry<String, Object> p : outParameters.entrySet()) {
            System.out.println("Name: " + p.getKey() + ", Value: " + p.getValue());
        }
        System.out.println("\nLoading time : " + (t1-t0) + " ms");
        System.out.println("Execution time : " + (t3-t2) + " ms");

    } catch (Exception exception) {
        exception.printStackTrace(System.err);
    }
}
}
```

Add code to initialize your Input parameter
Set your initialized input parameter
Add code to examine/print the output parameter

Decision engine for zRule Execution Server © 2012 IBM Corporation

This is an example of the code that is generated for you when testing your decision engine rule project with Java. The yellow highlighted areas show where you need to modify or add code. The top one is where you add the code to set up and initialize the ruleset input parameters. In the middle, you modify the code to reference the input parameters. At the bottom, the generated code prints out the output parameter. However, in most cases, you normally add code here to check the output parameter values or to print in a more readable fashion.


Differences from the classic rule engine

- Compilation and deployment of rules is different (as previously discussed)
- Decision engine does not support:
  – Update with refresh modifier in an action rule
  – Finders (replaces rule conditions with application specific code)
- Decision Validation Services (DVS) do not support the decision engine
  – Decision engine projects can still use DVS
  – If results from DVS and running with decision engine are not the same, it is a bug that should be reported to IBM
- The Java APIs for using the engines from Java are different
- In the Rule Execution Server Console, the archive content cannot be viewed

Decision engine for zRule Execution Server

© 2012 IBM Corporation

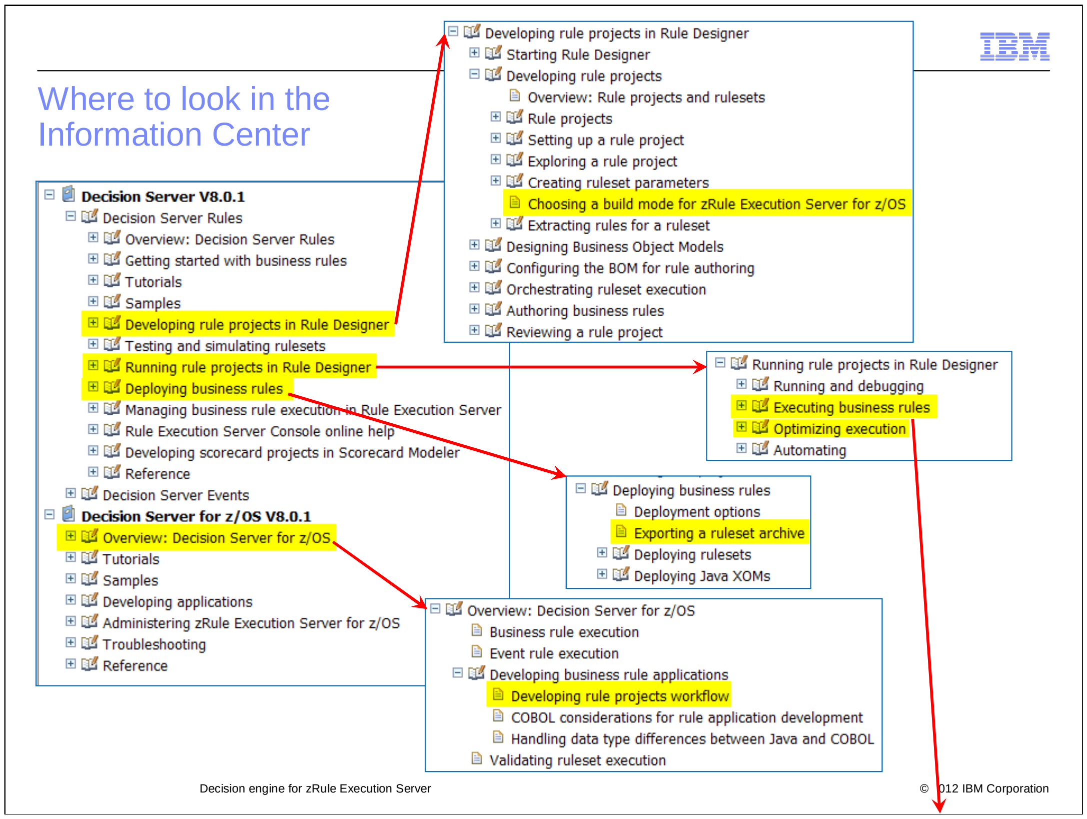
In addition to the architecture previously described for compilation and deployment of rules, there are other differences between the classic rule engine and the decision engine.

The decision engine supports neither the update with refresh modifier in an action rule, nor finders, the use of application specific code in place of rule conditions.

Decision validation services does not support the decision engine, but projects configured for the decision engine can still be tested with DVS. The results from DVS should be the same as results from rules deployed with the decision engine. If they are not, it is considered a bug and should be reported through your IBM support channels.

The Java API shown on previous slides is different than the Java API for testing classic rule engine projects.

Finally, the archive content cannot be viewed in the Rule Execution Server console.



Where to look in the Information Center

Decision Server V8.0.1
- Decision Server Rules
  - Overview: Decision Server Rules
  - Getting started with business rules
  - Tutorials
  - Samples
  - Developing rule projects in Rule Designer
  - Testing and simulating rulesets
  - Running rule projects in Rule Designer
  - Deploying business rules
  - Managing business rule execution in Rule Execution Server
  - Rule Execution Server Console online help
  - Developing scorecard projects in Scorecard Modeler
  - Reference
- Decision Server Events

Decision Server for z/OS V8.0.1
- Overview: Decision Server for z/OS
- Tutorials
- Samples
- Developing applications
- Administering zRule Execution Server for z/OS
- Troubleshooting
- Reference

Developing rule projects in Rule Designer
- Starting Rule Designer
- Developing rule projects
  - Overview: Rule projects and rulesets
  - Rule projects
  - Setting up a rule project
  - Exploring a rule project
  - Creating ruleset parameters
  - Choosing a build mode for zRule Execution Server for z/OS
  - Extracting rules for a ruleset
- Designing Business Object Models
- Configuring the BOM for rule authoring
- Orchestrating ruleset execution
- Authoring business rules
- Reviewing a rule project

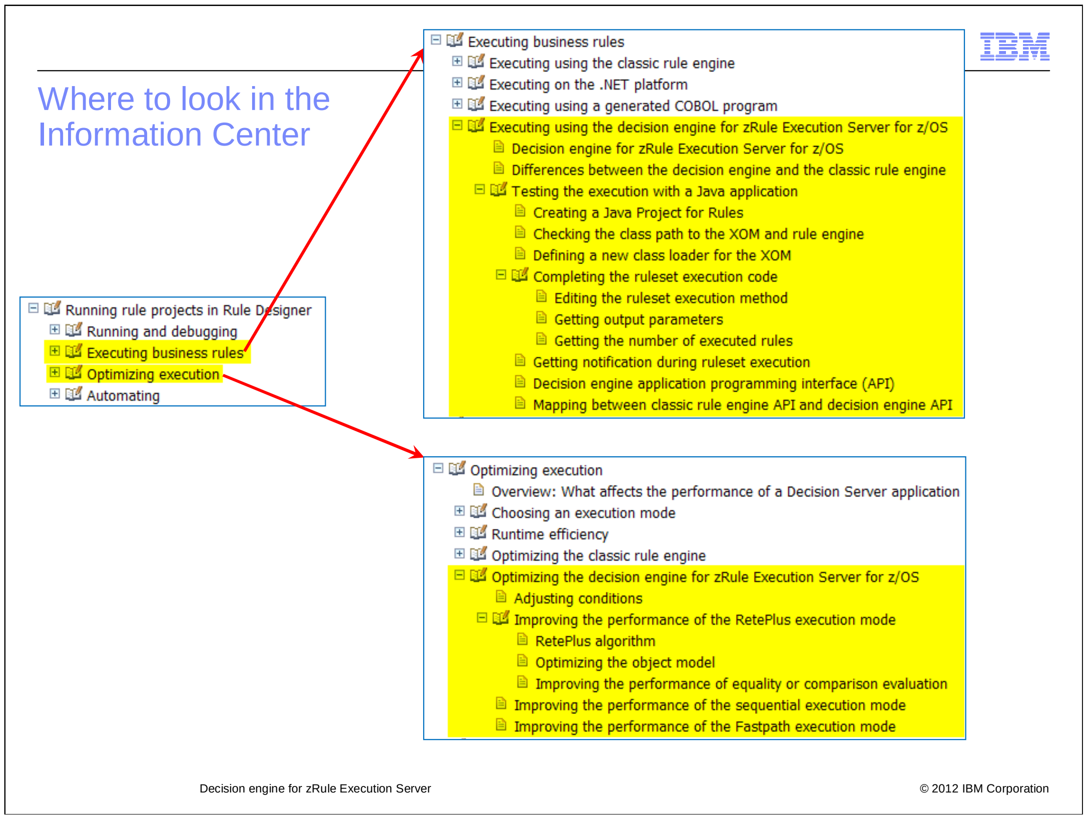
Running rule projects in Rule Designer
- Running and debugging
- Executing business rules
- Optimizing execution
- Automating

Deploying business rules
- Deployment options
- Exporting a ruleset archive
- Deploying rulesets
- Deploying Java XOMs

Overview: Decision Server for z/OS
- Business rule execution
- Event rule execution
- Developing business rule applications
  - Developing rule projects workflow
  - COBOL considerations for rule application development
  - Handling data type differences between Java and COBOL
- Validating ruleset execution

There are new sections in the Information Center describing the decision engine and there are other sections that are updated with decision engine specific information. This slide and the next provide you with a map to all the places in the Information Center that contain decision engine relevant content.

The arrow pointing off the bottom of the slide refers to the content shown on the next slide.



Where to look in the Information Center

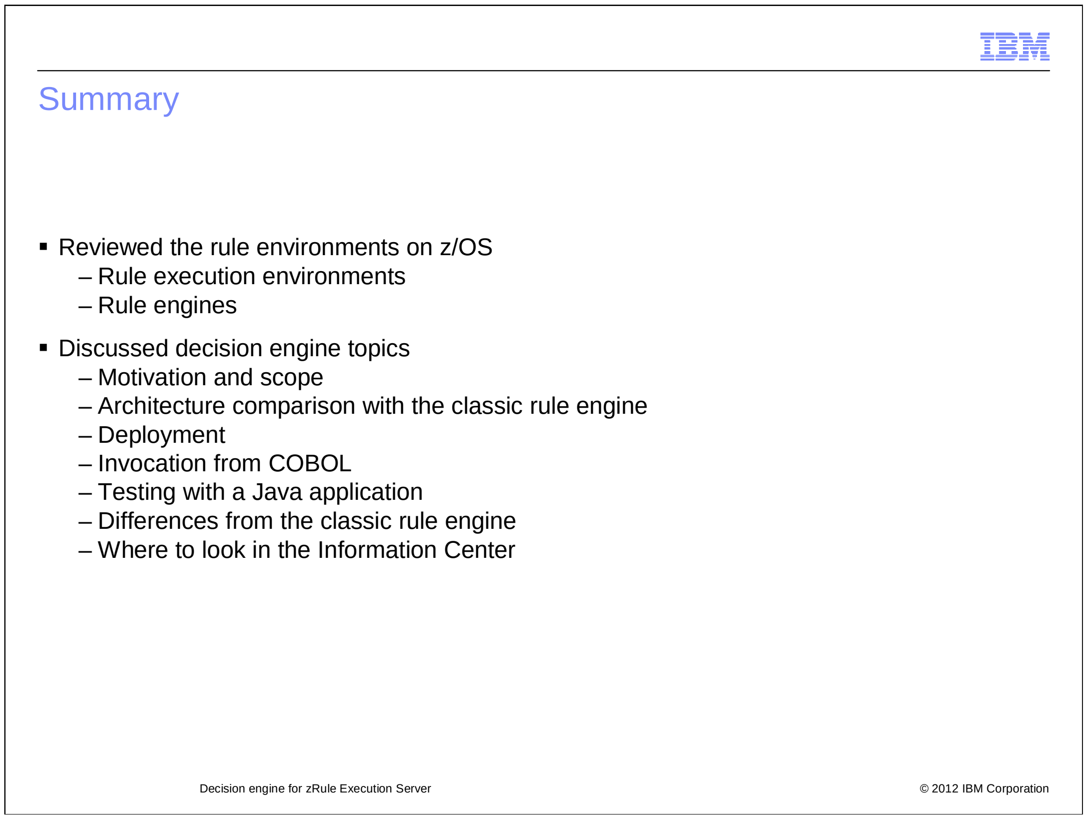
- Running rule projects in Rule Designer
  - Running and debugging
  - Executing business rules
  - Optimizing execution
  - Automating

- Executing business rules
  - Executing using the classic rule engine
  - Executing on the .NET platform
  - Executing using a generated COBOL program
  - Executing using the decision engine for zRule Execution Server for z/OS
    - Decision engine for zRule Execution Server for z/OS
    - Differences between the decision engine and the classic rule engine
    - Testing the execution with a Java application
      - Creating a Java Project for Rules
      - Checking the class path to the XOM and rule engine
      - Defining a new class loader for the XOM
      - Completing the ruleset execution code
        - Editing the ruleset execution method
        - Getting output parameters
        - Getting the number of executed rules
      - Getting notification during ruleset execution
      - Decision engine application programming interface (API)
      - Mapping between classic rule engine API and decision engine API

- Optimizing execution
  - Overview: What affects the performance of a Decision Server application
  - Choosing an execution mode
  - Runtime efficiency
  - Optimizing the classic rule engine
  - Optimizing the decision engine for zRule Execution Server for z/OS
    - Adjusting conditions
    - Improving the performance of the RetePlus execution mode
      - RetePlus algorithm
      - Optimizing the object model
      - Improving the performance of equality or comparison evaluation
    - Improving the performance of the sequential execution mode
    - Improving the performance of the Fastpath execution mode

Decision engine for zRule Execution Server

This slide is a continuation of the previous slide showing more Information Center content for the decision engine.

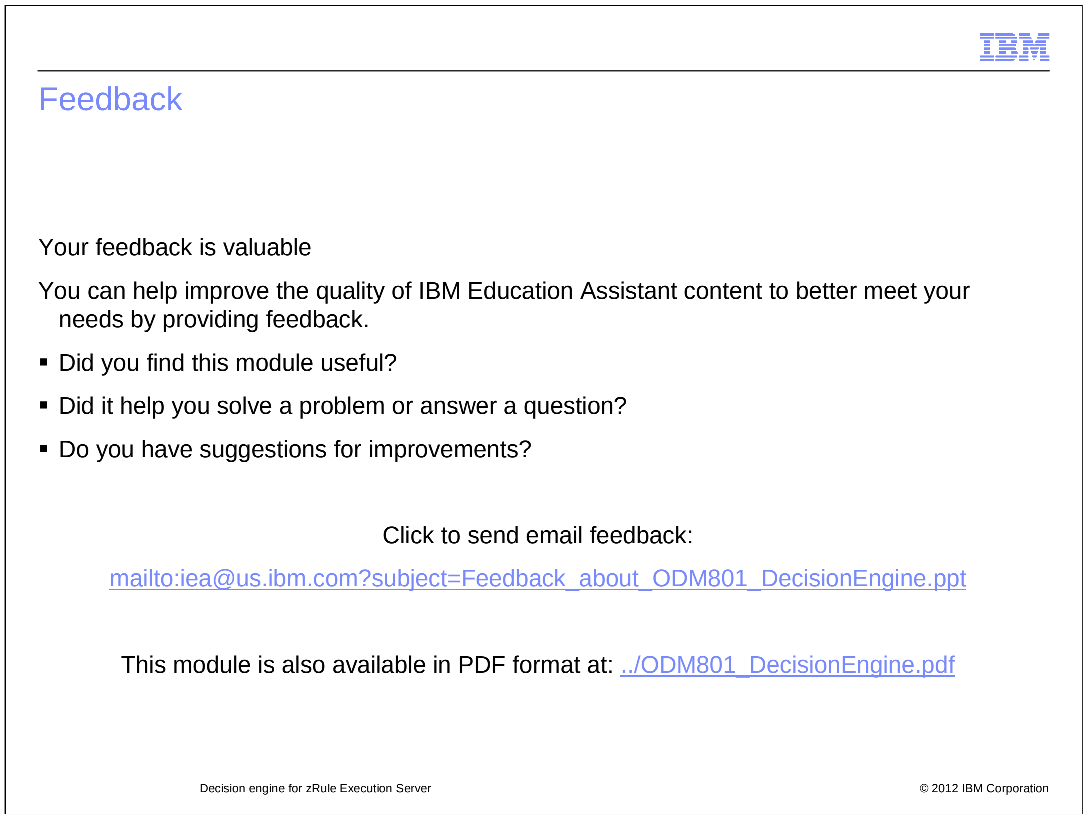


Summary

- Reviewed the rule environments on z/OS
  - Rule execution environments
  - Rule engines
- Discussed decision engine topics
  - Motivation and scope
  - Architecture comparison with the classic rule engine
  - Deployment
  - Invocation from COBOL
  - Testing with a Java application
  - Differences from the classic rule engine
  - Where to look in the Information Center

Decision engine for zRule Execution Server

In this presentation the rule environments for z/OS were reviewed. Then the decision engine specific topics were addressed, starting with the motivation for the decision engine and the scope to which it applies. The architecture was then compared to that of the classic rule engine. Following that, how rules using the decision engine are deployed, invoked from COBOL and tested with a Java application were presented. Some of the differences in the decision engine and classic rule engine were highlighted and a map was provided to help you find decision engine relevant material in the Information Center.



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_DecisionEngine.ppt

This module is also available in PDF format at: ../ODM801_DecisionEngine.pdf

Decision engine for zRule Execution Server

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

Trademarks, disclaimer, and copyright information