Redbooks

ibm.com/redbooks

# Flexible Decision Management with Business Rules on IBM z Systems

Mark Hiscock

Guy Hindle

Mike Johnson

Tim Wuthenow

James Taylor

David Griffiths

Graeme Everton

z Systems

IBM

Redbooks

International Technical Support Organization

# Flexible Decision Management with Business Rules on IBM z Systems

June 2015

SG24-8014-02

**Note:** Before using this information and the product it supports, read the information in “Notices” on page ix.

**Third Edition (June 2015)**

This edition applies to Version 8.7.1 IBM Operational Decision Manager for z/OS.

# Contents

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

# Trademarks

# Preface

The IBM® Operational Decision Manager product family provides value to organizations that want to improve the responsiveness and precision of automated decisions. This decision management platform on IBM z/OS® provides comprehensive automation and governance of operational decisions that are made within mainframe applications. These decisions can be shared with other cross-platform applications, providing true enterprise decision management.

This IBM Redbooks® publication makes the case for using Operational Decision Manager for z/OS and provides an overview of its components. It is aimed at IT architects, enterprise architects, and development managers looking to build rule-based solutions. Step-by-step guidance is provided about getting started with business rules by using a scenario-based approach. This book provides detailed guidelines for testing and simulation and describes advanced options for decision authoring. Finally, it describes and documents multiple runtime configuration options.

**Third edition:** This third edition, SG24-8014-02, of this IBM Redbooks publication updated the information presented in this book to reflect function available in IBM Operational Decision Manager for z/OS Version 8.7.1.

## Authors

This third edition of this IBM Redbooks publication was produced by a team working at the IBM International Technical Support Organization, Raleigh Center.

**Mark Hiscock** is the Operational Decision Management development team lead for z/OS based in Hursley, UK. He joined IBM in 1999 and holds a first class degree in Computer Science from the University of Portsmouth. He has over 10 years experience working in mainframe development on products such as: ODM, IBM WebSphere® MQ, IBM CICS®, IBM DB2®, WebSphere Message Broker, and IBM WebSphere Application Server.

**Guy Hindle** is the team lead for Operational Decision Manager on z/OS development based in the IBM Hursley development laboratory in the UK. He has worked in IBM for 20 years and in IT for over 25 years. Previous IBM roles include team lead, software development, test, and education roles on projects including application software development, middleware, and services engagements. Before joining IBM, he developed application generation tools for HR solutions and tools for corporate management accounting. His expertise is in usability, software application design, and database development.

**Mike Johnson** is a senior developer for Operational Decision Manager on z/OS in the IBM Hursley development laboratory in the UK. In this role, he also acts as an ambassador for Operational Decision Manager at various technical conferences. He has a number of patents and publications and was one of the authors of the first edition of this publication. Mike holds a Combined Honours degree in Computing and Statistics from Aston University. He has a particular interest in Ergonomic Design and has over 25 years working in integration software.

**Tim Wuthenow** is an IT Specialist for WebSphere on IBM z™ Systems. He holds a degree in Chemical Engineering from North Carolina State University. His area of expertise is in the configuration of WebSphere Operational Decision Management on z/OS and CICS. He developed numerous client demonstrations using the new capabilities of WebSphere Operational Decision Management within the CICS and z/OS environments. He was also one of the authors of the first edition of this publication.

**James Taylor** is an FVT lead for Operational Decision Manager on z/OS. Performing this work, he is often the first to test drive new function developed for the product on z series. During the 14 years he has worked for IBM (at the Hursley Laboratory, UK), he performed a diverse set of roles ranging from leading the documentation team for CICS Transaction Gateway and IBM TXSeries® to designing the user experience of IBM Worklight®, the IBM mobile development platform.

**David Griffiths** is a senior member of the Operational Decision Manager on z/OS development team based in the Hursley development laboratory. After obtaining a degree in Physics at Manchester University, David started programming in 1977 for Marconi Radar and among many other things, worked as a UNIX kernel developer, ported Java to NeXTSTEP and RiscOS before joining the team porting Java to IBM OS/390® at IBM in 1998.

**Graeme Everton** is a member of the development team for Operational Decision Manager on z/OS, based in the IBM Hursley development laboratory in the UK. He has worked for IBM for nine years and in IT for 20 years. Previously, he worked in Development Operation Infrastructure for IBM z Systems™ Build Team and before joining IBM, he worked as a Mobile Telecoms Technical Consultant, a University Computer Systems Manager, IT Support Engineer, and ran his own Mobile Computer Support company.

Thanks to the authors of the first and second editions of this book: Chris Backhouse, Andy Flatt, Fiona Crowther, Janet Wall, Sebastien Brunot, Pierre Feillet, Youngxin Pan, and Wand Wenjie. We thank them for their excellent work, much of which is still contained in this third edition.

The third edition of this IBM Redbooks publication project was led by:

Rufus P. Credle Jr., IBM Consulting IT Specialist and Information Developer
International Technical Support Organization, Raleigh Center

Thanks to the following person for her contribution to this project:

**Emi Nakamura** is an information developer for Operational Decision Manager. She was a translation coordinator for Operational Decision Manager before becoming an information developer two years ago. She worked on the product for more than 10 years as a translator, a translation coordinator, and an information developer.

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

**ibm.com**/redbooks/residencies.html

## Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  **ibm.com**/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`
- Follow us on Twitter:

  `https://twitter.com/ibmredbooks`
- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`
- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

# Part 1

# IBM z Systems with business rules

This part describes IBM Operational Decision Manager for z/OS and contains the following chapters:

1

# The case for IBM Operational Decision Manager

This chapter introduces the concept of operational decision management and describes using IBM Operational Decision Manager for z/OS to address the agility needs of enterprise CICS, IBM IMS™, and batch COBOL or PL/I applications.

The following topics are covered in this chapter:

## 1.1 What Operational Decision Manager is

Smarter business outcomes require the ability to quickly adapt to change. But corporate leadership in every industry is struggling to keep pace with change whether it is driven by regulatory, market, or customer forces. These changes directly affect the corporation's business policies and the business decisions that are required to consistently apply business policies.

**Business policy:** A *business policy* is a statement of guidelines, which is implemented via business decisions.

For example, a bank might have a lending policy stating, "Customers whose credit rating is above average are entitled to a discounted rate on their loan". The traditional application development lifecycle requires a business analyst to document the detailed requirements and to design and develop this policy into one or more business applications. Then, one or more developers take those requirements and code or embed the decisions into the various application programs. The application development is then followed by a lengthy testing process. Unfortunately, the decisions are now hidden in the code of one or more programs, and over time as additional changes are added to the business policy, the code becomes more complex, making changes and auditability difficult.

Decision management is an important capability for delivering agile business solutions. *Decision management* is the "business discipline, supported by software that enables organizations to automate, optimize, and govern repeatable business decisions improving the value of customer, partner, and internal interactions." Decision management is the tool to help corporations accelerate their reaction to the pace of the growing complexity of business changes.

Accurate real-time business decisions provide many benefits within an organization. Better decisions help companies identify opportunities for increased revenue and profitability, such as in marketing and sales. Better decisions also help companies enforce compliance with external and internal policies, such as in claims processing or eligibility determination. Finally, better decisions help companies manage and reduce risk, such as with fraud detection and credit approvals.

### 1.1.1 Common business decisions that require managing

There are, generally speaking, three types of decisions that are found in most businesses that would benefit from better management:

- Decisions that help increase revenue:
  - This type includes decisions that are used by marketing and sales to make targeted offers based on customer profiles, demographics, and analytical models.
  - For example: Is a customer eligible for a certain promotion or a cross-sell or up-sell opportunity? Should a store discount the price of a product at the end of the day?
- Decisions around consistency and compliance with regulations:
  - This type of decision can be found in all industries, such as financial, insurance, and government sectors.
  - For example: Are there prohibitions against a customer buying a certain quantity of a product? Is a customer eligible to make a certain purchase based on where she is located?

- Decisions that reduce and mitigate risk:
  - The third type of decision includes those decisions that help reduce and mitigate risk.
  - For example: Does the customer who just filled out a loan application online meet the criteria to be approved?

Businesses must make one or all of these types of decisions, many times a day, ensuring they are made correctly and consistently according to their business policies.

The objective of this book is to describe how Operational Decision Manager for z/OS can be used to implement decision management in the CICS, IMS, and batch COBOL or PL/I applications of an organization.

### 1.1.2 Where are most decisions made today?

The traditional approach to decision making requires a business analyst to understand the business policies and create a requirements document, which defines the decisions to be made. Then, one or more software developers take the requirements and code or embed the decisions into the various application programs that support the business. This is achieved by coding business rules into the chosen application programming language.

**Note:** Business decisions are made by evaluating one or more business rules.

The application development is then followed by a lengthy testing process before the new decisions become live in production.

Unfortunately, the decisions are now hidden in the code of one or more programs, and over time as additional changes are added to the business policy, the code becomes more complex, making it difficult to change, hard to visualize, and nearly impossible to manage. The decisions can change frequently or rarely and changing a program to change the decision, testing it, and getting it into production is not fast enough in today's business environment.

This scenario can be avoided by implementing a decision management solution, which takes the decisions out of code and places them in a central repository. This makes the decisions more flexible, visible, auditable, and manageable. This is illustrated in Figure 1-1.

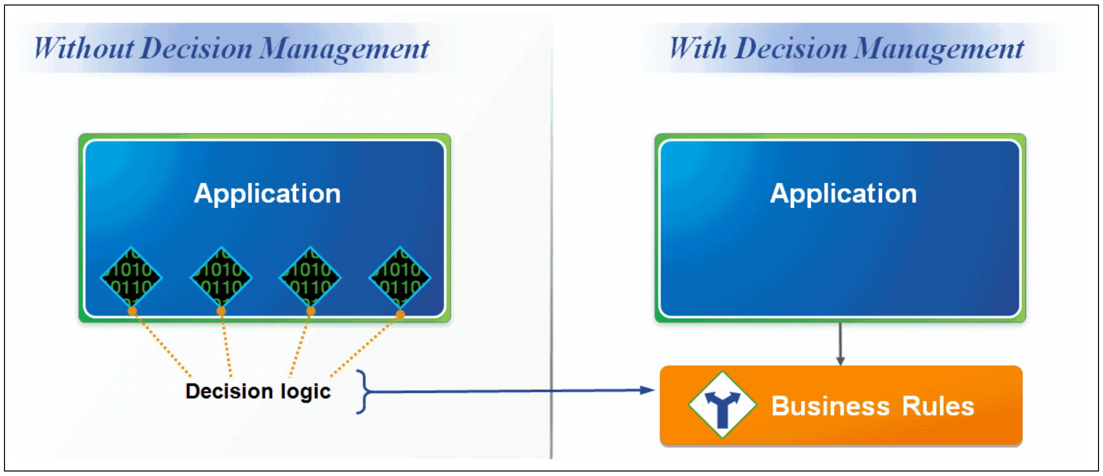


*Figure 1-1 The decision logic is moved out of code and into a decision manager*

Not all decisions are equal and some are more applicable for decision management than others. The following decisions are best suited:

- Those decisions that must be changed frequently to support the business

  Decision management avoids costly application code changes.

- Those decisions that are duplicated in multiple applications running on multiple platforms

  Decision management implements the decision once and stores it centrally, allowing it to be called from multiple applications.

- Those decisions that must be visible for business purposes

  Decision management allows decisions to be shared easily with lines of business or regulatory auditors.

The next section introduces IBM Operational Decision Manager for z/OS and explains how it can be used to implement a decision management solution on z/OS.

## 1.2 Operational Decision Manager for z/OS

Organizations embark on application modernization projects to enable their core IBM z Systems business applications to respond rapidly to emerging opportunities. First, these business applications must be understood in terms of the business decisions they implement and the effect of decision changes on key business processes.

Organizations can efficiently implement application modernization projects by incrementally externalizing their business decisions from COBOL or PL/I applications and moving them into a decision management system. Most companies begin using Operational Decision Manager with one or possibly two business decisions at a time. Taking an incremental approach with decision management in core business applications provides organizations with a return on investment (ROI) in the first phase of their projects. An incremental approach also avoids embarking on a lengthy, labor-intensive “rip and replace” project. It finally enables the team to understand the design and management techniques of decision management.

Operational Decision Manager combines the authoring, testing, and management of business rules that are required for implementing business decisions. Operational Decision Manager enables organizations to adapt incrementally the business decisions in their mainframe applications while avoiding lengthy application development cycles.

Operational Decision Manager offers the following features:

- A set of tools for business users, administrators, and developers to edit and manage rules
- A powerful decision engine to execute business decisions
- A robust decision repository to centrally host the business decisions
- An extensive library to define and extend the decision execution and management environment

Applying Operational Decision Manager to application modernization projects can incrementally address projects in the following areas:

- Effective application maintenance: z/OS development teams need to address their long list of maintenance projects for their core COBOL or PL/I business applications. If a maintenance project requires updates to the decisions that are implemented in a specific application, redesign those rules in Operational Decision Manager for enhanced ongoing management.

- Consolidating or restructuring existing applications: Most organizations have duplicate functionality in multiple applications, which causes a company to spend more time and resources maintaining applications than is necessary. Consolidation combines the same functionality into a single core business application. Using Operational Decision Manager technology for these modernization projects centralizes the business decisions that were previously duplicated.
- Sharing business decisions across applications and platforms. This area is an effective way to obtain a higher ROI. Operational Decision Manager for z/OS provides the tooling to design business decisions for your COBOL and PL/I applications that can be reused across z/OS and distributed platforms.

## 1.3 Where Operational Decision Manager for z/OS can be used

Operational Decision Manager for z/OS provides intelligent and responsive decision management for mainframe applications. It enables organizations to build solutions that can automate the decision response to transactional and process-oriented business systems.

The decisions are translated into business rules, which are the detailed conditions and actions that unambiguously enforce the business decision. The business rules state in detail the circumstances under which the decision is applicable and the actions that enforce it. Business decisions can be defined that result in new application behaviors, offering a quick and productive route to enhance the business responsiveness of CICS, IMS, and batch operations. Also, the business owner can be in control of the management of business decisions and at the same time good governance and change management can be enforced.

Designing, maintaining, testing, implementing, and managing business decisions within Operational Decision Manager provides the following benefits:

- A convenient communication channel between IT and business teams
- Easy implementation and reuse of business decisions across the enterprise
- Flexible options for progressive IT modernization

Operational Decision Manager for z/OS can benefit your organization in many ways:

- Customer relationship: By improving customer interaction and personalization:
  - Achieve finer-grained personalization in customer interactions. Business rules enable business users to implement more tailored promotions, pricing, risk models, and so on, therefore, increasing the precision and personalization of operational decisions.
  - Move decision making to the point of contact with customers and enable enterprises to deploy decisions at the contact point with customers and improve consistency in decisions about customers and customer interaction.
- Enterprise processes: By improving business alignment, compliance, and transparency:
  - Achieve high pass-through rates in process automation. Centrally managing business decisions enables you to streamline processes and helps you achieve higher levels of automation and higher pass-through rates by externalizing decisions and automating more complex decisions.
  - Maximize decisions for resources, risk, and value. Managed business decisions enable businesses to tie sources of insight (from historical data, predictive knowledge, simulation, and events) and decision automation capabilities to achieve consistently better business outcomes and maximize resources and value.

- Business agility and speed: By improving business-led agility and responsiveness:
  - Empower business users to manage and improve decisions. Managed business decisions provide an agile platform to enable business users to manage decisions and changes in a short time frame.
  - Shorten response time to changing market conditions.
  - Increase enterprise responsiveness to unforeseen events, as well as shortened response time and time-to-business due to higher levels of automation.

## 1.4 Who deploys Operational Decision Manager for z/OS?

For a team to be effective, it is necessary to have the right set of skills in, or available to the team, for consultation. As stated in the IBM Redpaper™ publication *Making Better Decisions using WebSphere Operational Decision Management*, REDP-4836, the responsibility for specifying and managing the business is considered from the point of view of the business roles:

- Business analysts are responsible for specifying how the business needs to behave, identifying key performance indicators (KPIs) that reflect how well the business is doing, and defining the processes and decision points that are needed to manage the business.
- Line-of-business (LOB) users are responsible for the day-to-day management of the business using the solutions. They are responsible for monitoring the KPIs and modifying the way that decisions are made in order to optimize the business. In a decision management solution, these roles have the responsibility for optimizing decisions to meet the business need.
- Users are responsible for using the solution, and need to consider the solution from a consumability and process-efficiency perspective. In many cases, the user role might be the subject of KPIs that the solution is designed to support.

The responsibility for the delivery and maintenance of these systems lies with the IT department. When embarking on a project using Operational Decision Manager technology, there are new and expanded project roles. There are several key roles to include in these projects:

- Application subject-matter expert (SME)

  An SME on the application that is being modernized is an essential member of the team. This individual provides awareness of the programming styles in use and an understanding of the role that the application serves. Ideally, this SME is aware of the application's programming history, including the original purpose and design and how the application changed over time.
- Enterprise architect

  The enterprise architect can provide valuable context for the application's role in the business decisions that are managed.
- Business rule analyst/rule designer

  This role understands the business decisions and rules for the new implementation. This person provides input to the new design of the rules. This role is normally combined with the business rule designer/implementer.
- Business object modeler

  This role defines the business object model for the target application and maps the COBOL or PL/I structures to business-friendly vocabulary.

- Business rule repository administrator

  This role is responsible for ensuring that the business object model and rules are defined consistently for all phases of the project and ensures that the rules can be shared across platforms.
- Business rule miner

  This role is optional. Normally, this person is the COBOL or PL/I developer, or a technical business analyst, responsible to “drive through the code” to identify the candidate business rules. This person enters them in the rule authoring user interface.

Operational Decision Manager for z/OS provides the ability for all the roles shown here to participate in the creation, maintenance, and execution of the business decisions.

In Chapter 2, “IBM Operational Decision Manager for z/OS” on page 11, the multiple rule authoring and execution capabilities offered by Operational Decision Manager for z/OS are explained. It highlights the power of using Operational Decision Manager for z/OS when adopting a business rules approach for application modernization on mainframe systems.

2

# IBM Operational Decision Manager for z/OS

This chapter provides an overview of how IBM Operational Decision Manager provides first class decision management capabilities on z/OS.

The following topics are covered in this chapter:

## 2.1 Operational Decision Manager for z/OS overview

Operational Decision Manager for z/OS provides a smarter way for dealing with business decisions. It is designed to help organizations gain more control and visibility over the business decisions that take place in their enterprise applications. Businesses that use Operational Decision Manager for z/OS will simplify their ability to create and make changes to decisions required by enterprise applications. This enables them to cut costs and cycle times, improve their agility and time to market, and enhance their visibility of business decisions and the governance of those decisions.

Operational Decision Manager for z/OS can deliver these advantages to enterprise users because it enables the separation of the decision logic from business application code and processes.

### 2.1.1 Operational Decision Manager for z/OS capabilities

The Operational Decision Manager product family provides value to organizations that want to improve the responsiveness and precision of automated decisions on z/OS and distributed applications. On z/OS, this decision management platform provides comprehensive automation and governance of the operational decisions that are made within mainframe applications.

Operational Decision Manager for z/OS consists of two orderable products, illustrated in Figure 2-1 on page 13, which together form a platform for the management and execution of business rules:

- *IBM Decision Center for z/OS* provides an integrated repository and management components for line-of-business (LOB) subject matter experts (SMEs) to directly participate in the governance of business rule-based decision logic. Through the capabilities of the Decision Center, business and IT functions can work collaboratively. They align the entire organization in the implementation of automated decisions and accelerate the maintenance lifecycle as they evolve, based on new external and internal requirements.

  Decision Center provides the following features:

  – Comprehensive decision governance, including role-based security, custom metadata, multiple branch release management, non-technical testing and simulation, and historical reporting.
  – Team collaboration through multiple user access for business users and integrated synchronization between IT and business user environments.

  Decision Center packaging includes these environments and tools:

  – Decision Center Enterprise console for advanced authoring.
  – Decision Center Business console for business users.
  – Decision Center repository for centrally storing rules.
  – Rule Solutions for Office for authoring rules in Microsoft Office.

  For additional information about Decision Center for z/OS, see 2.2, “Decision Center for z/OS” on page 14.

- IBM Decision Server for z/OS provides the runtime components to execute rule-based decision logic on mainframe systems. This product enables exact decisions to be made based on the context of each interaction.

With Decision Server for z/OS, an organization can process a business decision against hundreds or even thousands of business rules to determine how to respond within both front-end and back-end systems.

This product includes these components:

– Decision Server run times

  These run times are designed to handle the unique aspects of business rule execution. Decision Server for z/OS offers several mainframe runtime options for CICS, IMS, or batch applications. These options allow development teams to choose a deployment strategy that best fits their mainframe applications and architecture.

– Development tooling

  Rule Designer provides an application development environment for authoring business rules. It is used as the starting point to create a model on which the business rules are authored. Rule Designer is the Eclipse-based development toolkit for business rules. It is installed on a workstation.

For additional information about the runtime options that are available in Decision Server for z/OS, see 2.3, “Decision Server for z/OS” on page 15.

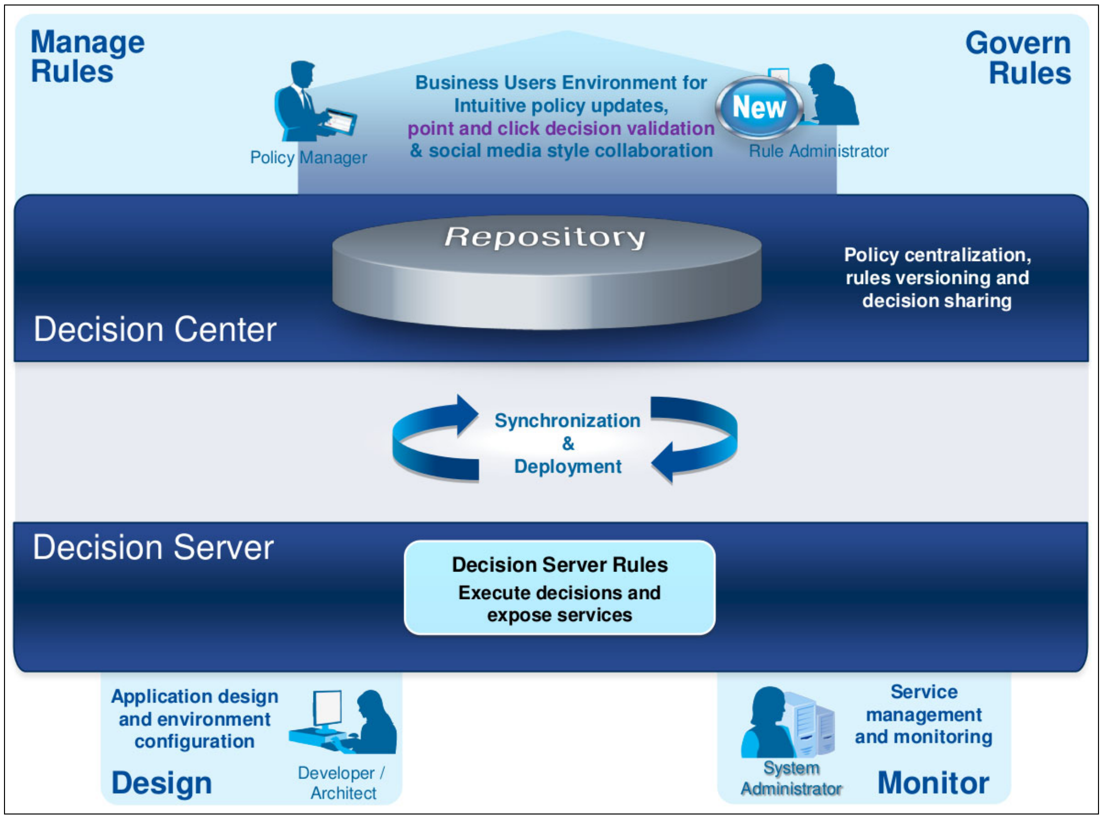

Manage Rules
Govern Rules
Business Users Environment for Intuitive policy updates, point and click decision validation & social media style collaboration
New
Policy Manager
Rule Administrator
Repository
Decision Center
Policy centralization, rules versioning and decision sharing
Synchronization & Deployment
Decision Server
Decision Server Rules
Execute decisions and expose services
Application design and environment configuration
Design
Developer / Architect
Service management and monitoring
System Administrator
Monitor

Figure 2-1 Overview of Operational Decision Manager for z/OS

## 2.2 Decision Center for z/OS

*Decision Center for z/OS* provides an integrated repository and the management components for LOB SMEs to directly participate in the governance of decision logic. Through the capabilities of Decision Center, business and IT functions can work collaboratively. Decision Center helps you to align the entire organization in the implementation of automated decisions. It helps you to accelerate the maintenance lifecycle as the automated decisions evolve based on new external and internal requirements.

### 2.2.1 Features

Decision Center is the central hub that coordinates the decision lifecycle across the business and IT parts of your organization. It provides the following features for business users to manage their decisions:

- Rule authoring

  Decision Center includes editors for authoring business rules and decision tables. These editors are made available via a web console.

- Rule synchronization between users and developers

  Synchronization is the key to collaborative work between business and IT users. You can adopt a developer-centric or a business user-centric approach to managing synchronization.

- Rule review and management

  In the Decision Center consoles, business users can run queries and publish reports on the content of their projects. Decision Center provides ways to customize how business users can view the items in their projects with smart folders. Business users can also manage releases and work with branches and baselines.

- Rule validation

  Decision Center provides tools for validating that decisions are implemented as expected. For business rules, Decision Center provides testing and simulation of rulesets.

- Rule deployment

  Following verification, you can deploy your decision logic as rule applications to the production system.

- Administration

  After you configure Decision Center, you perform several regular administrative tasks to provide optimum service to the business users.

Decision Center runs inside WebSphere Application Server for z/OS and can reside on z/OS or Linux for z Systems. It can be deployed, as well, in a distributed environment to edit the business rules. It can be configured to deploy those rules to a Decision Server running on z/OS.

### 2.2.2 Decision Center consoles

Operational Decision Manager for z/OS provides two consoles as web-based applications called the *Business console* and the *Enterprise console*. These are the primary methods for business users to edit and author business rules. The Business console is aimed at business subject matter experts who manage and govern lifecycle of decisions. Whereas the Enterprise console functionality is for advanced business analysts and administrators to fully develop, manage, and govern decisions. All decision-authoring assets accessed via these consoles are versioned and persisted in an underlying database repository as shown in Figure 2-2.

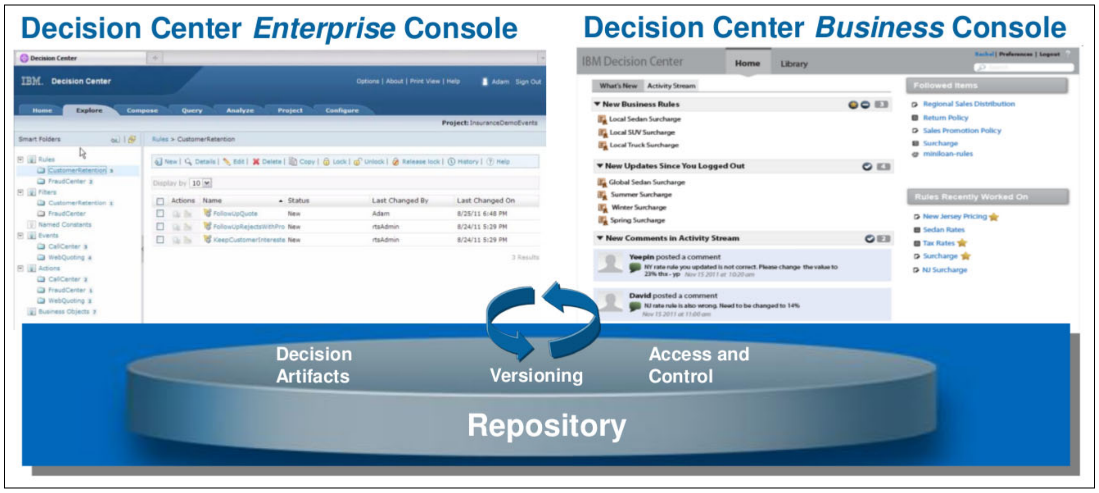

Decision Center *Enterprise* Console
Decision Center *Business* Console
Decision Artifacts
Versioning
Access and Control
Repository

*Figure 2-2 Operational Decision Manager for z/OS environment options*

For more information about the Decision Center consoles and how they are used in the full rule lifecycle, see Chapter 4, “Managing business decisions through the full lifecycle” on page 81.

## 2.3 Decision Server for z/OS

Decision Server for z/OS provides the ability to author business decisions based on COBOL copybook or PL/I include files. It then offers a number of z/OS based execution runtimes in which the business decisions can be executed from COBOL and PL/I applications. The following are key elements of Decision Server for z/OS:

- Business decision development

  To develop a business decision for z/OS, you design business rules independently from the application logic. Using *Rule Designer*, you develop rules and create a contract between the application and the rule execution. An application can call the rules in a number of ways using various execution options.

- Executing business decisions on z/OS

  You have a number of options to execute decisions on z/OS, which allow CICS, IMS, and batch COBOL or PL/I applications to call the Decision Server for z/OS.

- Validation of ruleset execution on z/OS

  To improve the business decisions, you can test the ruleset execution and simulate scenarios.

Decision Server on z/OS offers multiple execution environments with and without a WebSphere Application Server base. See 2.3.2, “Execution options for business decisions on z/OS” on page 16, which focuses on the execution environments of Decision Server.

## 2.3.1 Rule Designer

*Rule Designer* is used for the base rule authoring. This tooling is Eclipse-based and installed on a workstation. It cooperates with the Decision Center, which is installed on a distributed operating system or on WebSphere Application Server installed on z/OS.

This split-platform configuration enables users to implement full business rule management system (BRMS) functionality with the ability to define a business object model from COBOL or PL/I definitions, run and test rules on COBOL or PL/I data, and call the rule engine from existing CICS, IMS, and batch applications. Users who want to eliminate the duplication of application functionality on z/OS and distributed applications can consolidate applications and identify the rules that they can share between the two platforms.

The inclusion of COBOL and PL/I management in Rule Designer enables users to author rules and develop object models that can be shared with COBOL, PL/I, and Java applications. Developers import a COBOL or PL/I executable object model (XOM) from a copybook or include file and create a Java XOM along with a marshaller project to provide the mapping between Java and either COBOL or PL/I.

You do not have to re-engineer or rewrite an entire COBOL or PL/I application to start managing the business rules for your z/OS applications. You can base the scope of your rules on one, or a combination, of the following objects:

- A set of rules for a specific region, territory, or type of customer
- A process or subprocess within a z/OS application
- As a replacement for rules that might be hardcoded in your COBOL application

A rule-authoring environment is set up in the Rule Designer. Rule projects are created in the Rule perspective.

This rule project can be synchronized with Decision Center to be shared with business users who want to edit the rules. Alternatively, it can be deployed straight from Rule Designer to the Decision Server running on z/OS.

## 2.3.2 Execution options for business decisions on z/OS

When it comes to deployment, Operational Decision Manager for z/OS does not impose a rigid architectural or technical choice for business rule execution. Instead, it provides a set of options of which you can take advantage of, depending on your strategy, application, and architectural preferences.

Operational Decision Manager provides three options for running the Decision Server on z/OS:

- zRule Execution Server for z/OS (zRES)

  This option provides a native integration with existing COBOL or PL/I applications. A supplied stub program provides an API to directly execute decisions in the zRule Execution Server for z/OS. The zRES can run in three modes:

  - *Stand-alone mode* provides a zRES address space that can be invoked from existing COBOL or PL/I applications by way of the supplied API stub.
  - *Embedded mode* allows the zRES to run inside the COBOL or PL/I address space. It is accessed using the same API stub as the stand-alone server. This mode provides local access to the rule execution data.

  – In IBM CICS Transaction Server for z/OS V4.2 and higher, a *local execution option* uses the new Java virtual machine (JVM) server environment to host the zRES within the CICS region.

- Rule Execution Server on WebSphere Application Server for z/OS

  This option brings the full power of WebSphere Application Server for z/OS to the rule execution on z/OS. Full high availability and scalability are provided by the underlying application server, and the full suite of decision management services is available. Because of the ability to consume COBOL or PL/I data structures directly, the Rule Execution Server environment can be easily integrated with existing COBOL or PL/I applications.

- COBOL code generation

  This option is for clients who want to retain their existing application architecture and manage their business decisions within COBOL application code. The rules in a COBOL application can be incrementally migrated to a central business rule repository for external management, directly by business users. Then, the rules can be generated back into COBOL code to be inserted into and called directly from the application. This option can also be a first step toward the incremental modernization of applications.

  **Note:** This option does not support the generation of PL/I code.

All of the execution options are highlighted in Figure 2-3 and all provide the following decision management values:

- Reduced risk, disruption, cost, and time to implement change
- Better visibility and maintainability of decision logic
- Improved decision logic reuse across applications

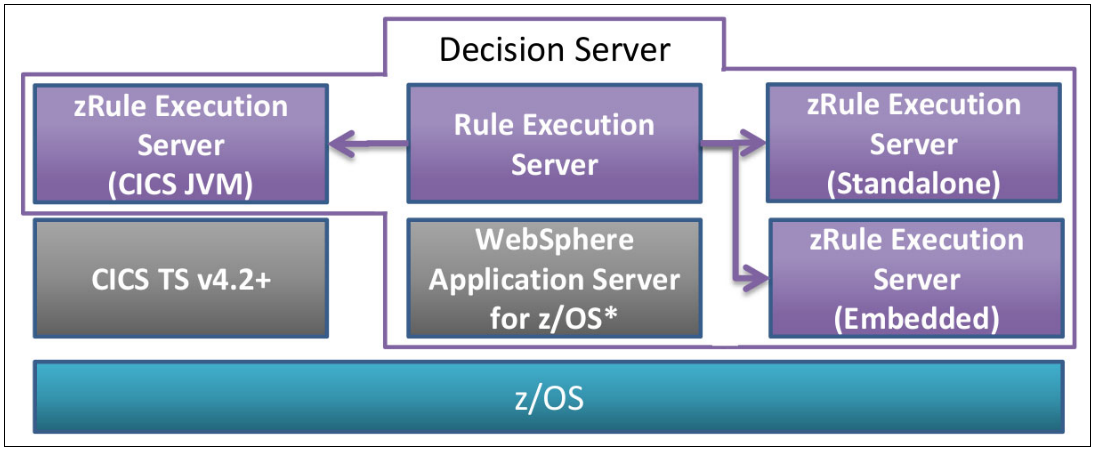

Decision Server
zRule Execution Server (CICS JVM)
Rule Execution Server
zRule Execution Server (Standalone)
CICS TS v4.2+
WebSphere Application Server for z/OS*
zRule Execution Server (Embedded)
z/OS

*Figure 2-3 Operational Decision Manager for z/OS rule execution options*

**Note:** A limited use entitlement for WebSphere Application Server for z/OS is included with Decision Server for z/OS for running the Rule Execution Server.

A feature comparison between zRule Execution Server for z/OS, Rule Execution Server on WebSphere Application Server for z/OS, and COBOL code generation is shown in Table 2-1.

*Table 2-1 zRule Execution Server for z/OS feature comparison*

| **Feature** | **zRule Execution Server for z/OS (zRES)** | **Rule Execution Server (RES) on WebSphere Application Server for z/OS** | **COBOL code generation** |
|---|---|---|---|
| Execution from Java | Yes | Yes | No |
| Execution from COBOL and PL/I | Yes | Yes, through WebSphere Optimized Local Adapter | Yes |
| OOTB COBOL and PL/I marshalling | Yes | Yes | N/A |
| Testing | Yes | Yes | No |
| Simulation | Yes | Yes | No |
| Hosted transparent decision services | No | Yes | No |

Figure 2-4 shows the detailed environment options to execute business decisions from COBOL or PL/I applications in CICS, IMS, and batch.

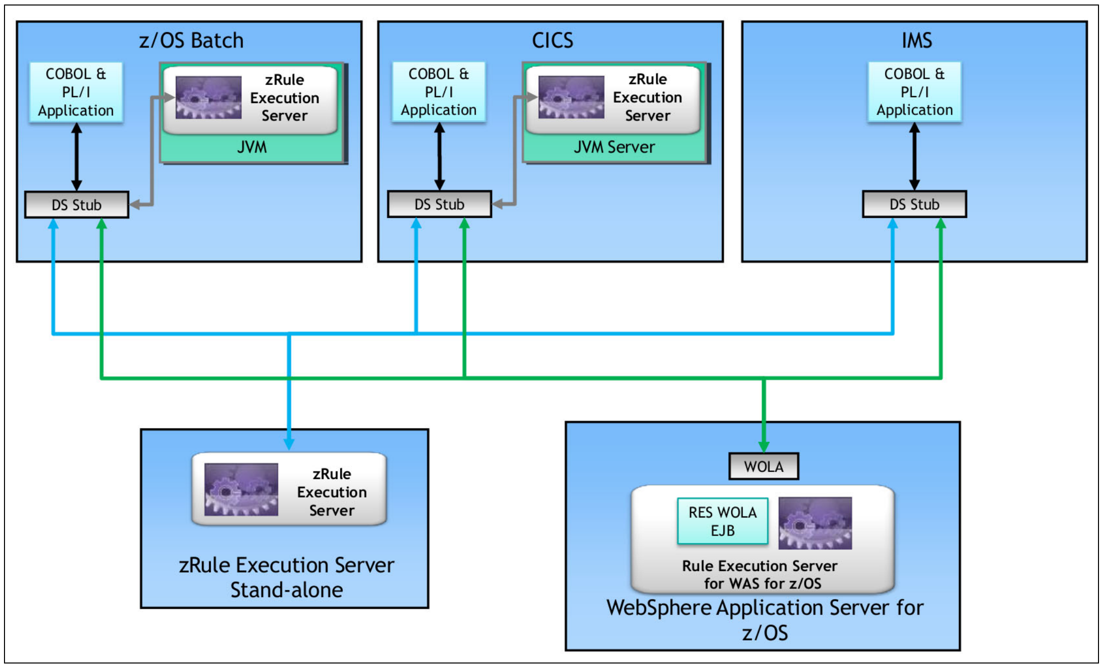


*Figure 2-4 All the Decision Server for z/OS execution options*

Some of the features such as running the zRule Execution Server for z/OS inside the batch address space are new since version 8.0.1. Section 2.4, "New in Operational Decision Manager Version 8.7.1" on page 19 explains all the new features since version 8.0.1.

## 2.4 New in Operational Decision Manager Version 8.7.1

The previous edition of this IBM Redbooks publication, *Flexible Decision Automation for Your zEnterprise with Business Rules and Events*, SG24-8014, was based on version 8.0.1 of Operational Decision Manager for z/OS. This third edition is based on version 8.7.1 and therefore the new capabilities that were included in versions 8.5.0, 8.5.1, 8.6.0, 8.7.0, and 8.7.1 are described in this section.

### 2.4.1 XOM and marshaller deployment through Decision Center

Since version 8.7.1, the XOM and marshaller JAR files associated with a Decision Service project can now be published to Decision Center. This allows a user in the Business Console to deploy the Decision Service, with the XOM and the marshaller, so that the Decision Service can be executed after a single deployment.

Previously, the XOM and the marshaller were required to be deployed separately using Rule Designer, through the RES Console or via a scripted mechanism.

### 2.4.2 Embedded mode

The zRule Execution Server for z/OS can run inside the batch address space since version 8.6.0. In this mode, rule execution requests access data locally in the address space, which improves performance for long running batch jobs. This is because no cross address space data copying is required as everything runs in the local address space.

See Chapter 9, “zRule Execution Server for z/OS embedded server” on page 147 for more information about the new embedded mode.

### 2.4.3 Stand-alone console address space

Since version 8.5.1, the Rule Execution Server (RES) console has been decoupled from the zRule Execution Server address space. The RES console can now be run and managed as its own address space. The HBRMODE runtime variable has been updated to include a CONSOLE mode, which allows the zRule Execution Server to run the RES console in a stand-alone address space.

Control statements can now be used to create a topology, which includes a single console and multiple zRule Execution Servers

### 2.4.4 PL/I support

The PL/I language is fully compatible with Operational Decision Manager for z/OS as of version 8.5.1. This means that PL/I include files can be imported to generate the Java executable object model, which is then used as the start point for the rule project.

Starting in version 8.6.0, existing Java based rule projects can be PL/I enabled, meaning that a PL/I include file can be generated to allow PL/I programs to call existing rule projects.

### 2.4.5 Decision engine across the product

The *decision engine* is designed to improve the overall performance of rule execution. The decision engine compiles rule artifacts into an archive that contains code that is ready to execute. The ruleset loading in the engine is faster because no code is parsed or interpreted at run time.

The decision engine is now supported across all of the Decision Server for z/OS execution options. It is also supported on distributed platforms since Operational Decision Manager version 8.5.1.

### 2.4.6 Business console

The *Business console* is aimed at business subject matter experts who manage and govern lifecycle of decisions. It is a separate interface to the Enterprise console, which provides a simpler experience for business users. It provides a modern web interface for managing and editing rules and testing, simulating, and deploying them.

For more information about the Business console, see Chapter 4, “Managing business decisions through the full lifecycle” on page 81.

### 2.4.7 IMS preinitialization routine

IMS message processing regions (MPRs) can gain performance benefits from using the preinitialization routine now supplied by Operational Decision Manager for z/OS since version 8.5.1. This routine issues the HBRCONN call to connect to the zRES at MPR start time. The IMS MPR applications must still issue an HBRCONN and HBRDISC API call, but they do not perform the full connect and disconnect routines.

### 2.4.8 Extended COBOL and PL/I data types

Since version 8.5.1, both COBOL and PL/I data types have been extended to include new data types.

#### PL/I support

The BIT string in PL/I is now supported when you generate a PL/I XOM from a PL/I include file. The single-bit BIT is mapped to the Java boolean type, and the multiple-bit BIT is mapped to the Java BitSet class.

#### COBOL support

Char, Enum, and Map data types in Java are now supported when you generate a COBOL copybook from an existing Java BOM. They can be mapped to COBOL data types.

### 2.4.9 Enhanced execution monitoring

Since version 8.6.0, the System Management Facilities (SMF) can be used to collect execution data, such as the number of decisions that are executed by your application at run time.

The SMF records are written by the rule execution environment, whether it is running in the zRule Execution Server for z/OS, a CICS rule-owning region, or Rule Execution Server on WebSphere Application Server for z/OS.

The SMF records are written as type 120 subtype 100 records. See Chapter 16, “Configuring Operational Decision Manager to collect execution data using SMF” on page 213 for more information about the records, their usage, and the data they contain.

## 2.4.10 Liberty support

Since version 8.7.0, the WebSphere Application Server Liberty profile for z/OS can now be used to host the ODM Rule Execution Server console and the Hosted Transparent Decision Service.

The WebSphere Application Server Liberty profile for z/OS provides a lightweight alternative to the traditional WebSphere Application Server for z/OS for running ODM.

For more information, see Chapter 12, “Liberty Application Server on IBM z/OS” on page 173.

## 2.4.11 COBOL code generation deprecation

Instead of using generated COBOL subprograms, the preferred mechanism for rule execution on z/OS is to deploy RuleApps to a zRule Execution Server for z/OS instance. If you are using the COBOL code generation feature in Rules for COBOL, you can use the COBOL Generation Project Migration wizard to migrate your rule project to a zRule Execution Server for z/OS compatible rule project.

However, if you are continuing to use COBOL code generation, be aware that in version 8.5.0, the COBOL code generation user interface is not displayed in Rule Designer by default. To display the COBOL code generation user interface in Rule Designer, stop Rule Designer, and add the following parameter in the file <*InstallDir*>/eclipse.ini:

```
-Dcobol.code.gen=true
```

Save the file, and restart Rule Designer to enable the COBOL code generation feature.

3

# Getting started with business rules

This chapter describes and demonstrates the use of business rules on IBM z Systems.

The following topics are covered in this chapter:

- 3.1, “Overview of the example used in this chapter” on page 24
- 3.2, “Getting started from a COBOL copybook” on page 25
- 3.3, “Getting started from an existing rule project” on page 67

## 3.1 Overview of the example used in this chapter

This section provides an overview of the business scenario and the related models that are used in this chapter.

### 3.1.1 Business scenario

The business scenario that is used in this book describes a fictitious auto insurance company that has a solution in place for validating an insurance application for its customers. The company wants to manage and share this validation logic with other business applications on z Systems.

The status of the in-place application and how a company wants the business rules to be executed on z Systems determine the approach that is chosen. This chapter uses the following typical approaches:

- Getting started from a COBOL copybook (see 3.2, “Getting started from a COBOL copybook” on page 25)
- Getting started from an existing Java rule project (see 3.3, “Getting started from an existing rule project” on page 67)

### 3.1.2 Business model

The business model used in this chapter represents an insurance quote application. This model is simplified as one insurance quote request and one insurance quote response.

#### Insurance quote request

The insurance quote request includes the following information:

- Driver

  This information includes personal information about the driver to assess the risk, such as age, address, and license status.

- Vehicle

  This information includes the make, model, year, and vehicle identification number, together with a categorization of the vehicle type and a vehicle value. The insurance discount policies differ based on the vehicle type.

#### Insurance quote response

The response to an insurance request includes the following information:

- The validation status of this insurance quote
- The validation message of this insurance quote
- The pricing and discount information of this insurance quote

### 3.1.3 Scenario rule model

The rules are designed to validate a customer’s eligibility for the quote application. These sets of rules validate the customer’s age or accident history and provide the validation result and possible reasons.

Both scenarios use the following rules:

- A maximum or minimum age rule, called *MaxiMinimumAge,* validates that the age of customer is between lower and upper age limits.
- A number of accidents rule, called *NumberOfAccidents*, validates that the number of accidents of the customer is below the set upper limits.

### 3.1.4 Project structure of a business rule on z/OS

This section illustrates a common project structure for business rule execution on zRule Execution Server for z/OS (zRES). The following typical artifacts are used in zRule Execution Server for z/OS projects:

- The *rule project*, which is used to design, debug, and manage the business rule
- The *Java Execution Module (Java XOM)*, which is used to create a rule project and, which is deployed as a Java archive (JAR) resource for rule execution at run time
- The *marshaller*, which handles the conversion between COBOL or PL/I data items and Java data items at run time

  **Important:** The marshaller is a generated JAR file artifact. Never modify it yourself.

- The *RuleApp project*, which is used to deploy business rules to the runtime configuration of the zRule Execution Server for z/OS

**Note:** The discussion of Decision Service projects is not included here. However, the Decision Service project is now the default project type in version 8.7.1 and should be used in place of the standard rule project.

## 3.2 Getting started from a COBOL copybook

This section includes detailed instructions about creating a rule application on z/OS that is started from a COBOL copybook. These instructions also apply to rule applications, which are built from a PL/I include file.

### 3.2.1 Scenario overview

In this scenario, an insurance company has a large COBOL application that runs on z/OS. This COBOL application validates insurance applications. The company wants to manage the logic codes that are scattered throughout the COBOL application and share them with other business applications on z Systems. The company decides to migrate and manage the business logic as a business rule application on z Systems.

In an actual scenario, you identify or create a COBOL copybook that contains the COBOL data items that are required for the business rules. Example 3-1 on page 26 shows a sample COBOL copybook (INSDEMO), which is designed for the first rule application on z Systems in this scenario.

*Example 3-1 Sample COBOL copybook [INSDEMO.cpy]*

```
01  REQUEST.
      05  DRIVER.
    10  FIRST-NAME            PIC  X(20).
    10  LAST-NAME             PIC  X(20).
        10  ZIPCODE               PIC  X(8).
        10  HOUSE-NUM             PIC  9(8).
        10  AGE                   PIC  9(2) USAGE COMP-3.
        10  LIC-DATE              PIC  X(8).
        10  LIC-STATUS            PIC  X.
        10  NUMBER-ACCIDENTS      PIC  99.
      05  VEHICLE.
        10  VEC-ID                PIC  X(15).
        10  MAKE                  PIC  X(20).
        10  MODEL                 PIC  X(20).
        10  VEC-VALUE             USAGE COMP-1.
        10  VEC-TYPE              PIC  X(2).
            88 SUV     VALUE 'SU'.
            88 SEDAN   VALUE 'SD'.
            88 PICKUP  VALUE 'PU'.
   01  RESPONSE.
      05  APPROVED                PIC  X.
      05  BASE-PRICE              USAGE COMP-2.
      05  DIS-PRICE               USAGE COMP-2.
      05  MSG-COUNT               PIC 9(5)  VALUE 0.
      05  MESSAGES                PIC  X(100)
                                  OCCURS 0 TO 100 TIMES
                                  DEPENDING ON MSG-COUNT.
```

**Additional resources:** You can find the `INSDEMO.cpy` copybook file in the additional information that is included in this book in the `code/Chapter3/CopybookBased` directory. See Appendix C, “Additional material” on page 239.

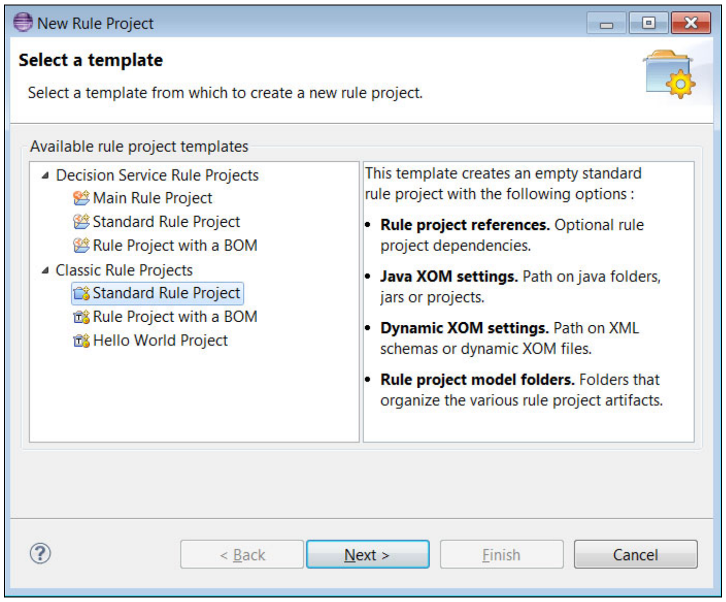
## 3.2.2 Creating a rule project

You can create a rule project in Rule Designer. A rule project enables you to manage, build, and debug the items that make up the business logic of your application.

In this scenario, we cover classic rules only.

Follow these steps to create the rule project in Rule Designer:

1. Click **File** → **New** → **Rule Project**. Select **Classic Rule Project** → **Standard Rule Project**, as shown in Figure 3-1 and click **Next**.

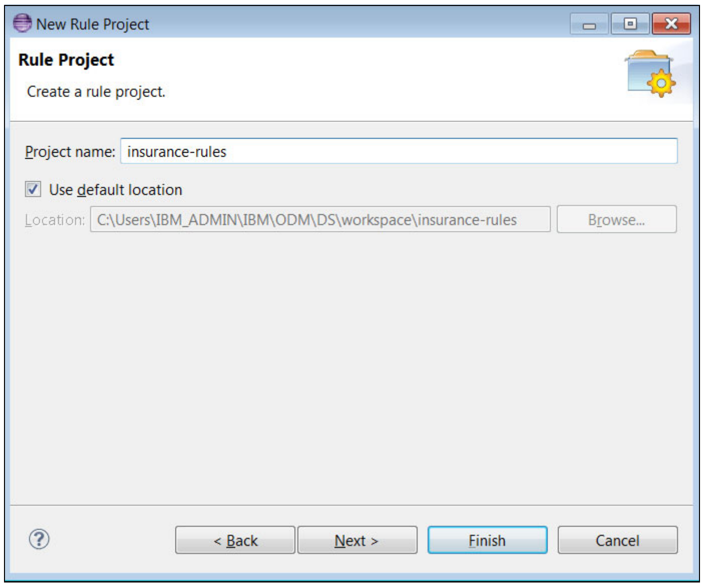


This template creates an empty standard rule project with the following options :

- **Rule project references.** Optional rule project dependencies.
- **Java XOM settings.** Path on java folders, jars or projects.
- **Dynamic XOM settings.** Path on XML schemas or dynamic XOM files.
- **Rule project model folders.** Folders that organize the various rule project artifacts.

*Figure 3-1 Selecting classic rule standard rule project*

2. In the next New Rule Project dialog, enter `insurance-rules` into the **Project name** field as shown in Figure 3-2. Click **Finish**.



Project name: insurance-rules

☑ Use default location

Location: C:\Users\IBM_ADMIN\IBM\ODM\DS\workspace\insurance-rules

*Figure 3-2 Creating a rule project*

The new rule project is created in the Rule Designer, as shown in Figure 3-3. For now, the rule project contains only empty folders.

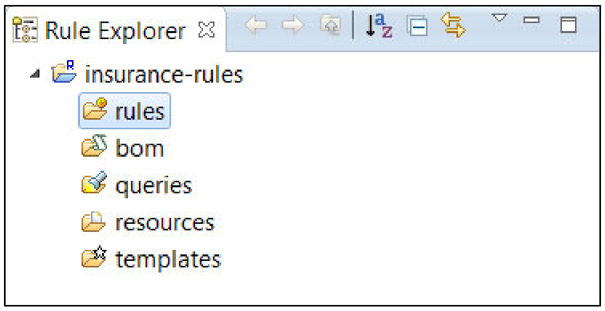


*Figure 3-3 New rule project in the Rule Explorer view*

## 3.2.3 Creating COBOL XOM from a COBOL copybook

To execute rules in a COBOL application, you generate the COBOL XOM from a COBOL copybook. A COBOL XOM provides the necessary COBOL to Java mapping so that you can create and execute your rules from a COBOL application.

To use the Rule Project Map to guide you through the COBOL XOM generation, follow these steps:

1. Select the **rules** folder in the newly created rule project (highlighted in Figure 3-3).
2. In the Design part of the Rule Project Map tab, click **Import XOM** (Figure 3-4).

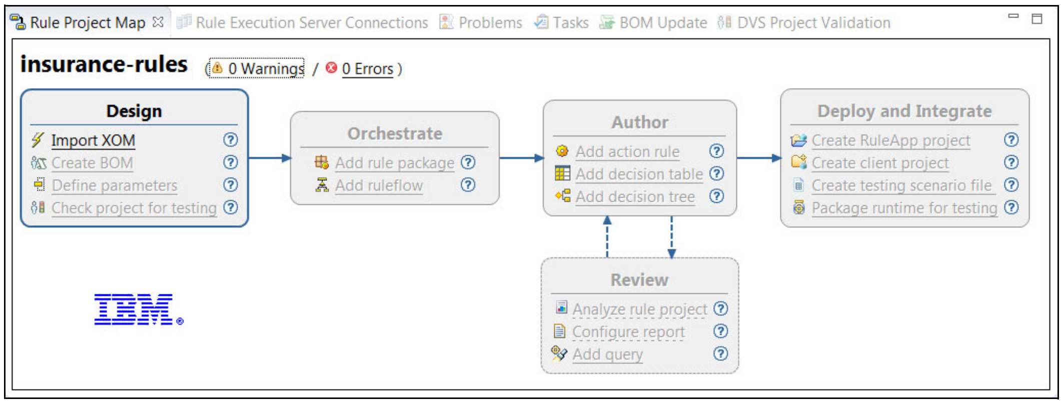
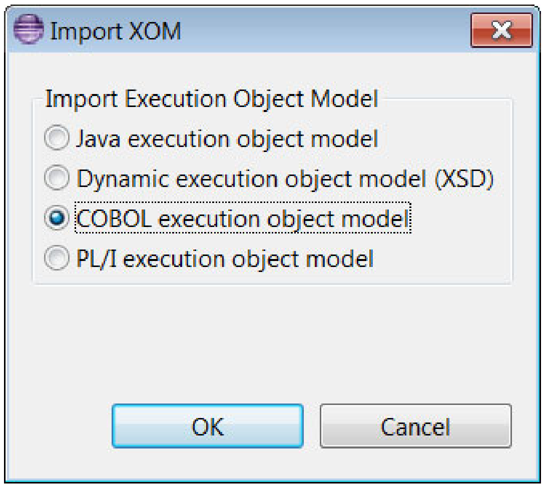


*Figure 3-4 Importing a XOM into a rule project map*

3. In the Import XOM dialog, select **COBOL execution object model** (Figure 3-5). Click **OK**.

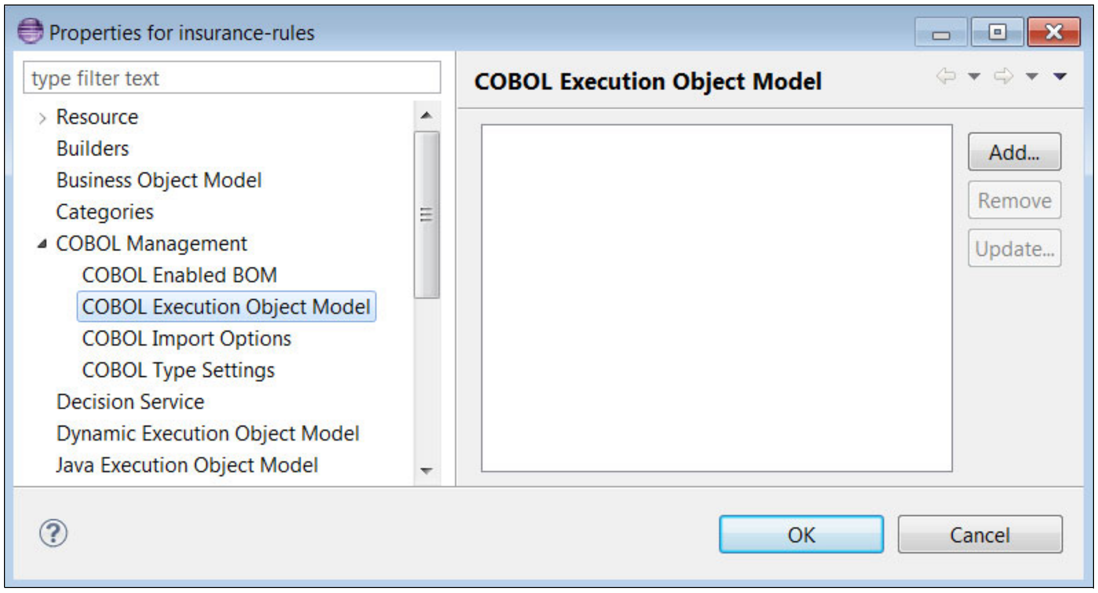

Import XOM

Import Execution Object Model

- Java execution object model
- Dynamic execution object model (XSD)
- [✓ COBOL execution object model]
- PL/I execution object model

*Figure 3-5 Selecting the COBOL execution object model*

4. On the Properties for insurance-rules dialog, click **Add** to add a COBOL Execution Object Model (Figure 3-6).


Properties for insurance-rules

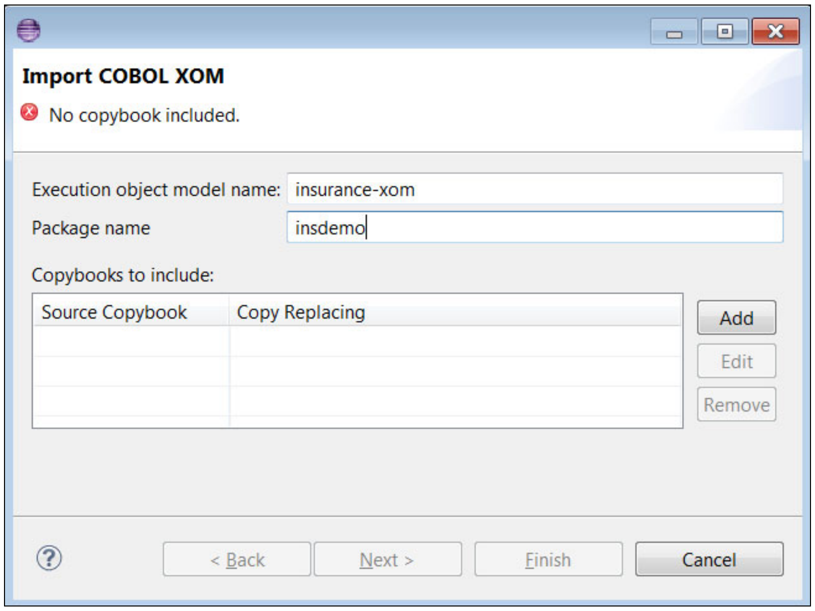
type filter text

- Resource
- Builders
- Business Object Model
- Categories
- COBOL Management
  - COBOL Enabled BOM
  - [✓ COBOL Execution Object Model]
  - COBOL Import Options
  - COBOL Type Settings
- Decision Service
- Dynamic Execution Object Model
- Java Execution Object Model

COBOL Execution Object Model

*Figure 3-6 Adding a COBOL Execution Object Model*

5. On the Import COBOL XOM dialog, in the Execution object model name field, enter `insurance-xom`, as shown in Figure 3-7. Click **Add**.


Import COBOL XOM

No copybook included.

Execution object model name: insurance-xom

Package name: insdemo

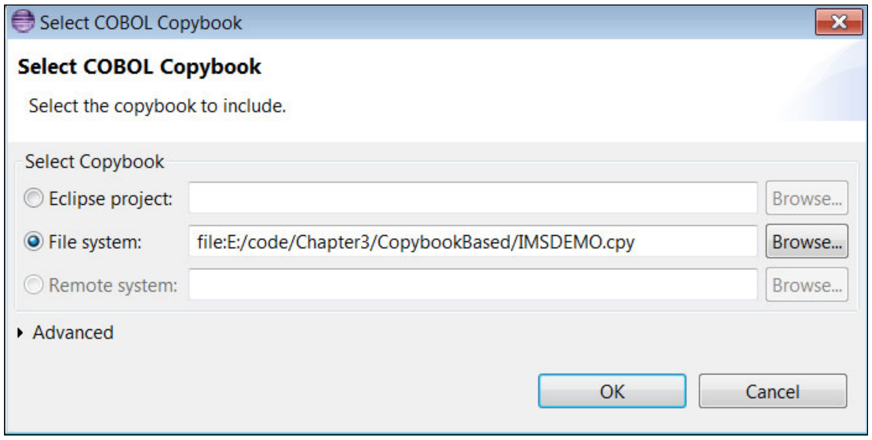
Copybooks to include:

| Source Copybook | Copy Replacing |
|---|---|

*Figure 3-7 Importing the COBOL XOM*

6. On the Select COBOL Copybook dialog, select a COBOL copybook by using one of the **Browse** buttons and select the **INSDEMO.cpy** copybook (Figure 3-8). Click **OK**.

> **Additional resources:** You can find the `INSDEMO.cpy` copybook file in the additional information that is included in this book in the `Chapter3/CopybookBased` directory. See Appendix C, "Additional material" on page 239.


Select COBOL Copybook

Select COBOL Copybook

Select the copybook to include.

Select Copybook

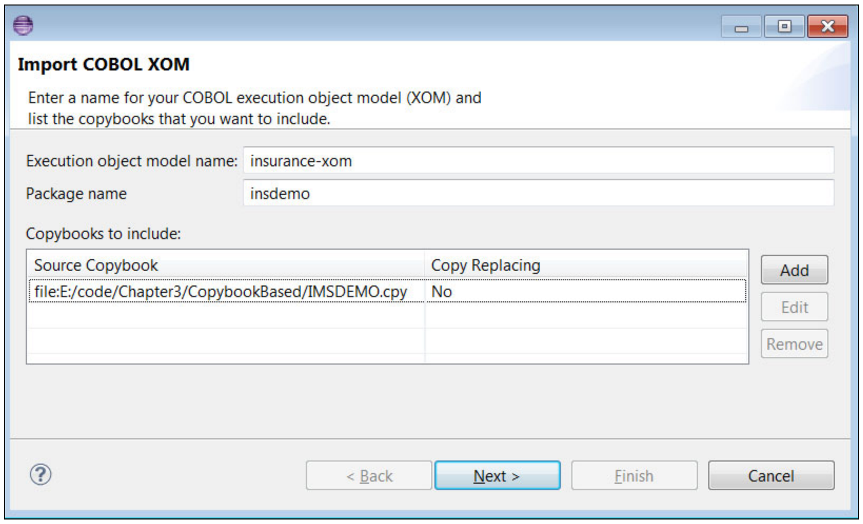
Eclipse project:

File system: file:E:/code/Chapter3/CopybookBased/IMSDEMO.cpy

Remote system:

*Figure 3-8 Selecting the COBOL copybook*

7. On the resulting Import COBOL XOM dialog (Figure 3-9), click **Next**.



Enter a name for your COBOL execution object model (XOM) and list the copybooks that you want to include.

Execution object model name: insurance-xom

Package name: insdemo

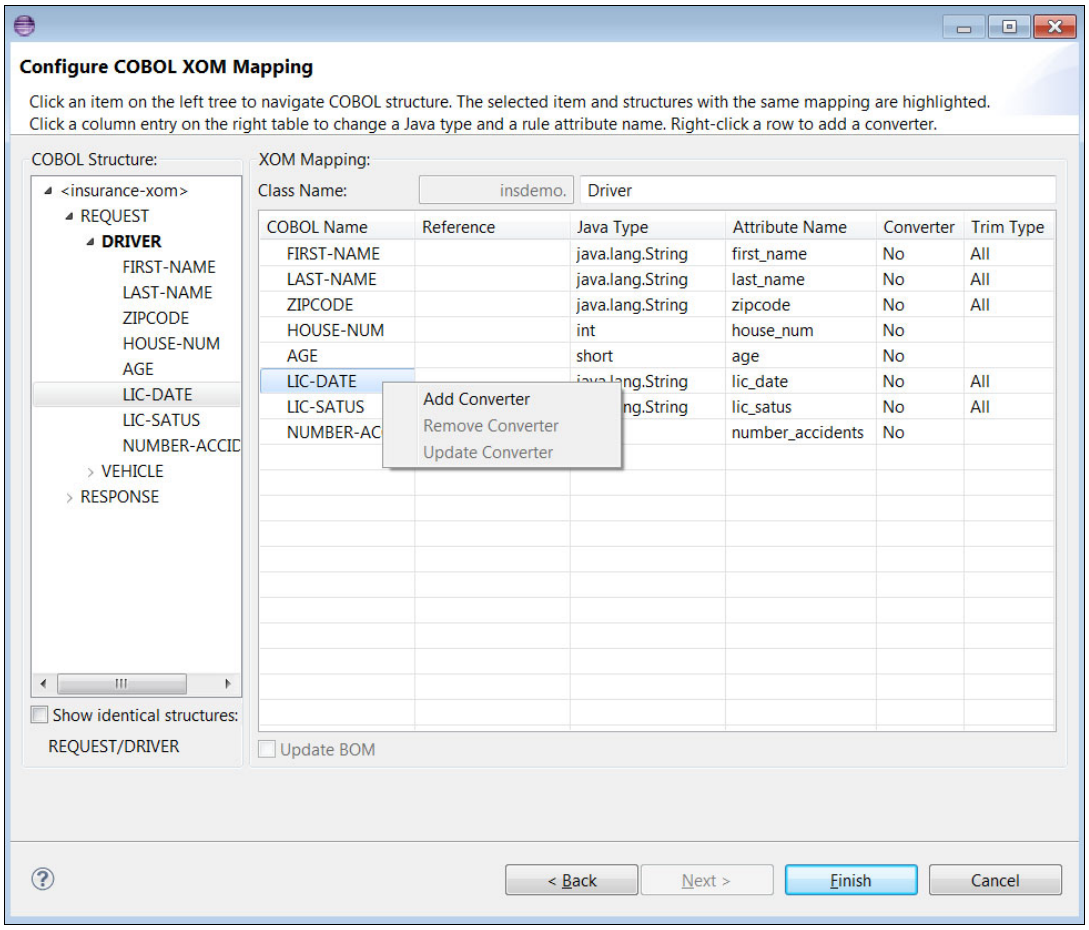
Copybooks to include:

| Source Copybook | Copy Replacing |
|---|---|
| file:E:/code/Chapter3/CopybookBased/IMSDEMO.cpy | No |

*Figure 3-9 Importing the selected COBOL copybook*

8. On the Configure COBOL XOM Mapping dialog, use the type converter to map two COBOL string items (PIC X) to a Java Date and a Boolean. Perform the following steps to change LIC-DATE from type String to Date by using the type converter:

   a. Expand the **<insurance-xom>** item. Expand the **REQUEST** item. Expand the **DRIVER** item. Then, right-click the row that contains the **LIC-DATE** data item in the **XOM Mapping** table, and click **Add Converter**, as shown in Figure 3-10 on page 32.



**Configure COBOL XOM Mapping**

Click an item on the left tree to navigate COBOL structure. The selected item and structures with the same mapping are highlighted.
Click a column entry on the right table to change a Java type and a rule attribute name. Right-click a row to add a converter.

COBOL Structure:

- <insurance-xom>
  - REQUEST
    - **DRIVER**
      - FIRST-NAME
      - LAST-NAME
      - ZIPCODE
      - HOUSE-NUM
      - AGE
      - LIC-DATE
      - LIC-SATUS
      - NUMBER-ACCID
    - VEHICLE
  - RESPONSE

Show identical structures:

REQUEST/DRIVER

XOM Mapping:

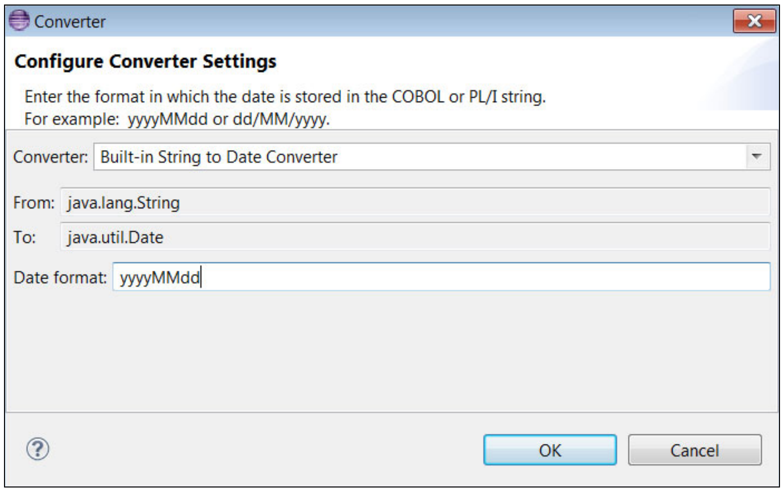
Class Name: insdemo. Driver

| COBOL Name | Reference | Java Type | Attribute Name | Converter | Trim Type |
|---|---|---|---|---|---|
| FIRST-NAME | | java.lang.String | first_name | No | All |
| LAST-NAME | | java.lang.String | last_name | No | All |
| ZIPCODE | | java.lang.String | zipcode | No | All |
| HOUSE-NUM | | int | house_num | No | |
| AGE | | short | age | No | |
| LIC-DATE | | java.lang.String | lic_date | No | All |
| LIC-SATUS | | ng.String | lic_satus | No | All |
| NUMBER-AC | | | number_accidents | No | |

Figure 3-10 Adding a converter for LIC-DATE item

b. On the Configure Converter Settings dialog, select **Built-in String to Date Converter**. Then, for the Date format field, enter `yyyyMMdd`, as shown in Figure 3-11. Click **OK**.

**LIC-DATE:** Use `yyyyMMdd` here to parse the LIC-DATE item value, such as `20110908`.



Converter

**Configure Converter Settings**

Enter the format in which the date is stored in the COBOL or PL/I string.
For example: yyyyMMdd or dd/MM/yyyy.

Converter: Built-in String to Date Converter

From: java.lang.String

To: java.util.Date

Date format: yyyyMMdd

Figure 3-11 Configuring the date converter

9. Change APPROVED from type String to Boolean, by using the type converter:

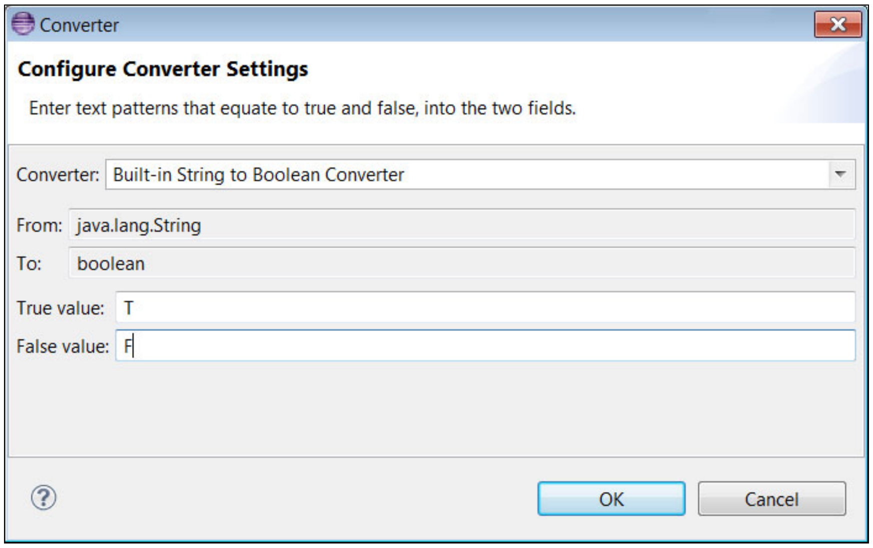
   a. Expand the RESPONSE item. Then, right-click the row that contains the APPROVED data item and click **Add Converter**.

   b. On the Configure Transform settings dialog, select **Built-in String to Boolean Converter**.

   c. For the True value field, type `T` and for the False value field, type `F`, as shown in Figure 3-12.

   d. Click **OK**.

**Values:** The T and F values are COBOL values that represent True and False. You can also customize these values as Y/N, YES/NO, and so on.



**Configure Converter Settings**

Enter text patterns that equate to true and false, into the two fields.

Converter: Built-in String to Boolean Converter

From: java.lang.String

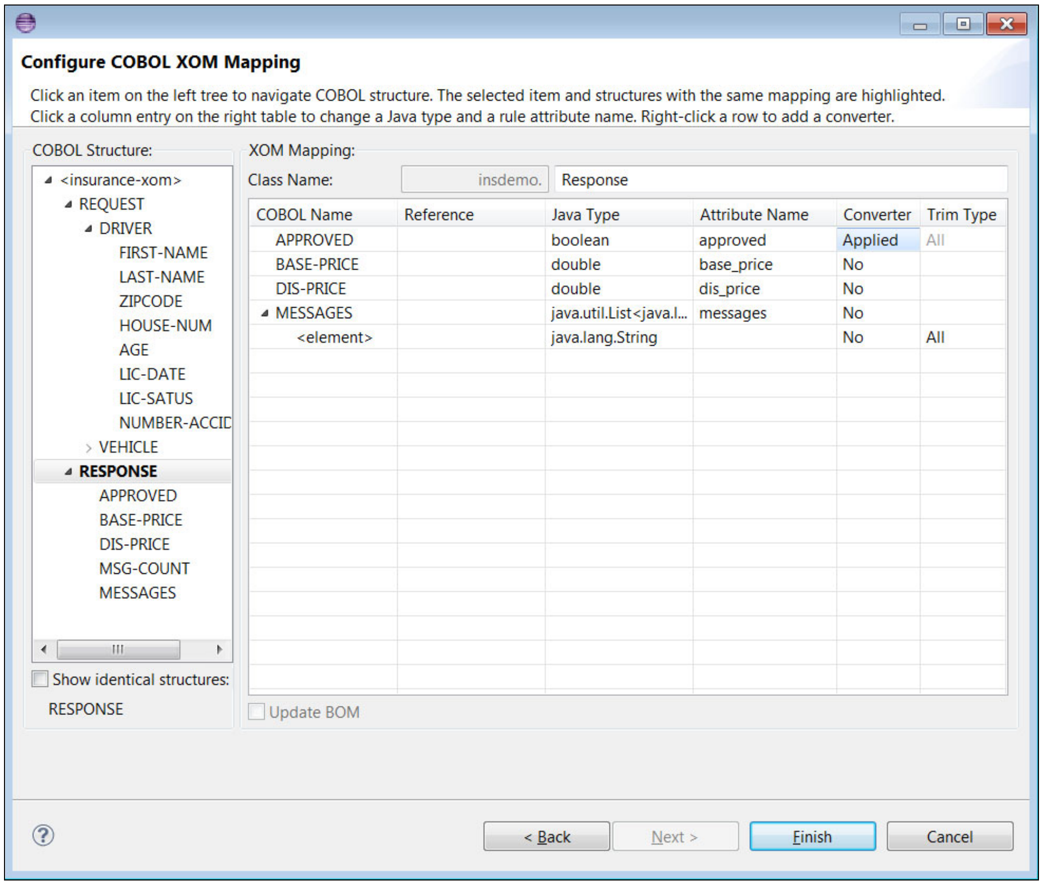
To: boolean

True value: T

False value: F

*Figure 3-12 Configuring the Boolean converter*

10. Click **Finish** to create the COBOL XOM (Figure 3-13).



**Configure COBOL XOM Mapping**

Click an item on the left tree to navigate COBOL structure. The selected item and structures with the same mapping are highlighted.
Click a column entry on the right table to change a Java type and a rule attribute name. Right-click a row to add a converter.

COBOL Structure:

- <insurance-xom>
  - REQUEST
    - DRIVER
      - FIRST-NAME
      - LAST-NAME
      - ZIPCODE
      - HOUSE-NUM
      - AGE
      - LIC-DATE
      - LIC-SATUS
      - NUMBER-ACCID
    - VEHICLE
  - **RESPONSE**
    - APPROVED
    - BASE-PRICE
    - DIS-PRICE
    - MSG-COUNT
    - MESSAGES

Show identical structures:

RESPONSE

XOM Mapping:

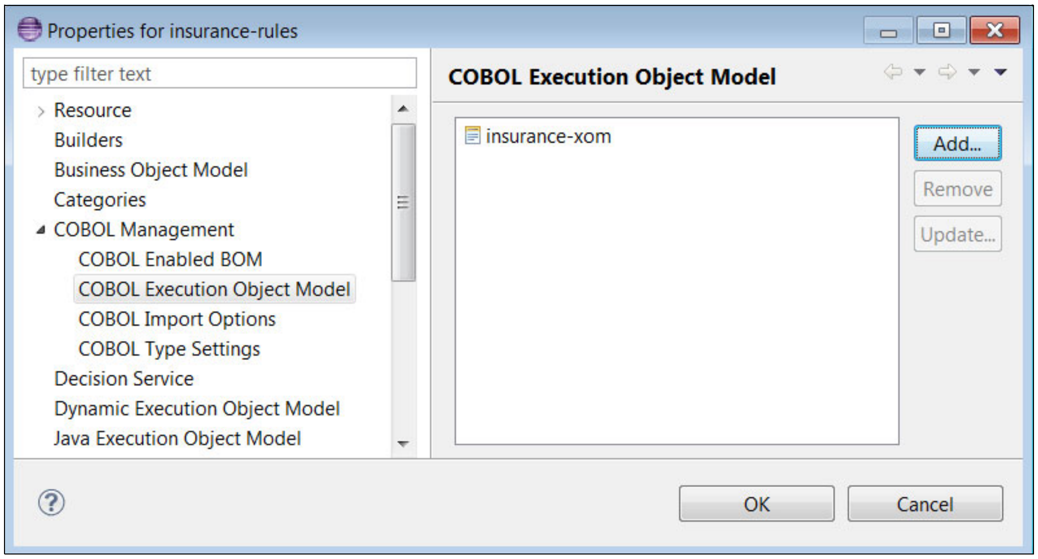
Class Name: insdemo. Response

| COBOL Name | Reference | Java Type | Attribute Name | Converter | Trim Type |
|---|---|---|---|---|---|
| APPROVED | | boolean | approved | Applied | All |
| BASE-PRICE | | double | base_price | No | |
| DIS-PRICE | | double | dis_price | No | |
| MESSAGES | | java.util.List<java.l... | messages | No | |
| <element> | | java.lang.String | | No | All |

Update BOM

*Figure 3-13 Finishing the data type configuration*

11. Click **OK** to close the Properties window (Figure 3-14).



Properties for insurance-rules

type filter text

- Resource
- Builders
- Business Object Model
- Categories
- COBOL Management
  - COBOL Enabled BOM
  - COBOL Execution Object Model
  - COBOL Import Options
  - COBOL Type Settings
- Decision Service
- Dynamic Execution Object Model
- Java Execution Object Model

**COBOL Execution Object Model**

insurance-xom

*Figure 3-14 Generating the COBOL XOM*

12. The following artifacts are created, as shown in Figure 3-15:

- Java XOM project: `insurance-xom`
- COBOL XOM:
  - Configuration file: `insurance-xom.xml`
  - Marshaller: `insurance-xom-xmarshaller.jar`

**Important:** The `insurance-xom` project and COBOL XOM files are generated artifacts. Do *not* change these artifacts manually.

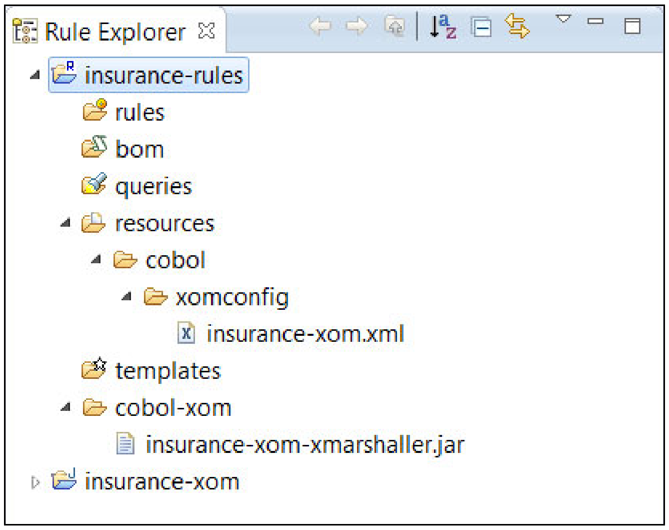


*Figure 3-15 COBOL XOM artifacts*

### 3.2.4 Creating a business object model from the Java XOM

The *business object model (BOM)* is a business layer that is used to author business rules. This section describes how to create a BOM in Rule Designer that is based on the Java XOM that you created in 3.2.3, “Creating COBOL XOM from a COBOL copybook” on page 28.

Follow these steps to create a BOM from the COBOL XOM:

1. In the Design part of the Rule Project Map tab, click **Create BOM**, as shown in Figure 3-16.

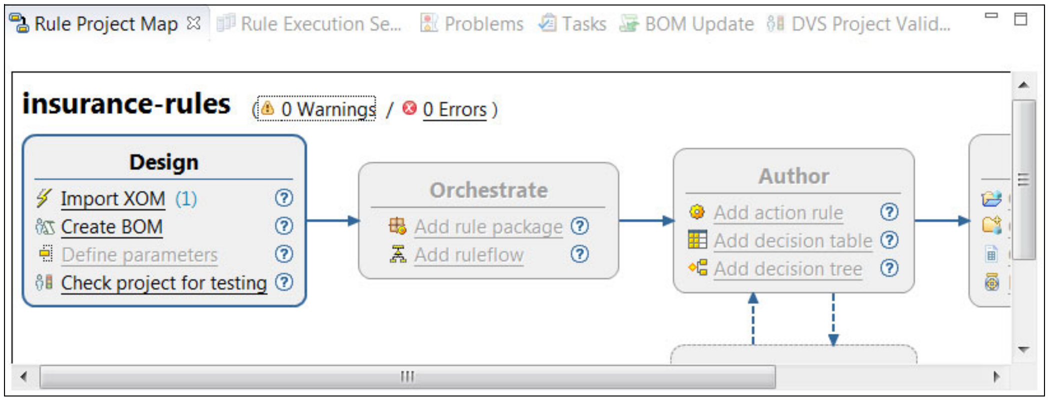
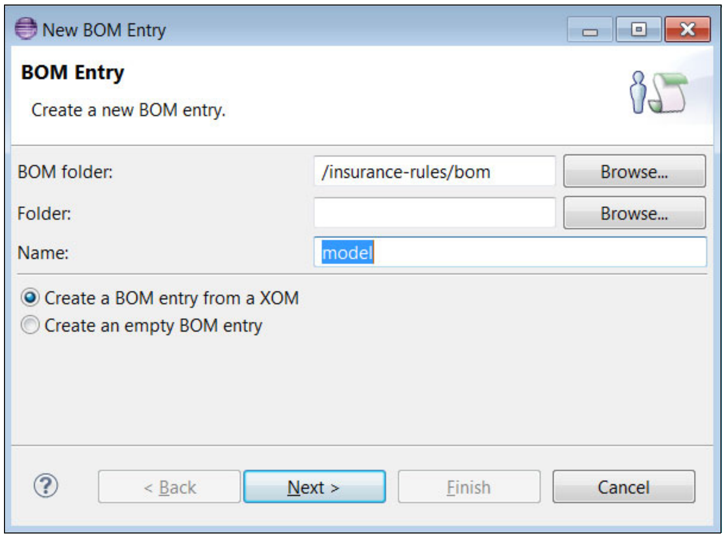


*Figure 3-16 Select “Create BOM” from the Rule Project Map*

2. In the New BOM Entry dialog, in the Name field, accept the default name for the BOM entry. In this scenario, the default name is `model`. Ensure that the **Create a BOM entry from a XOM** option is selected (Figure 3-17), and click **Next**.



BOM folder: /insurance-rules/bom
Folder:
Name: model

(●) Create a BOM entry from a XOM
( ) Create an empty BOM entry

Figure 3-17   Creating a BOM entry from a Java XOM

3. On the BOM Entry dialog, in the Choose a XOM entry field, click **Browse XOM**. On the Browse XOM dialog, select **insurance-xom**, as shown in Figure 3-18, and click **OK**.

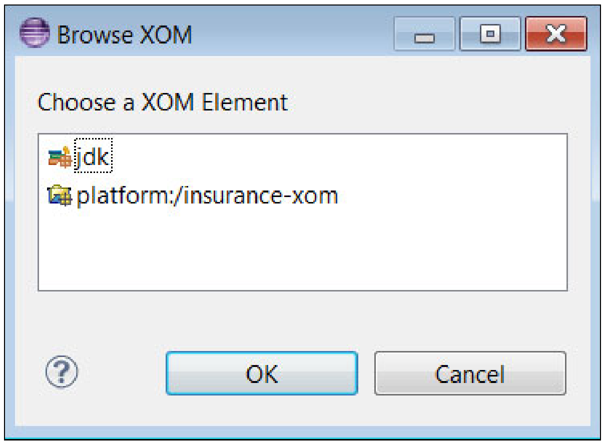
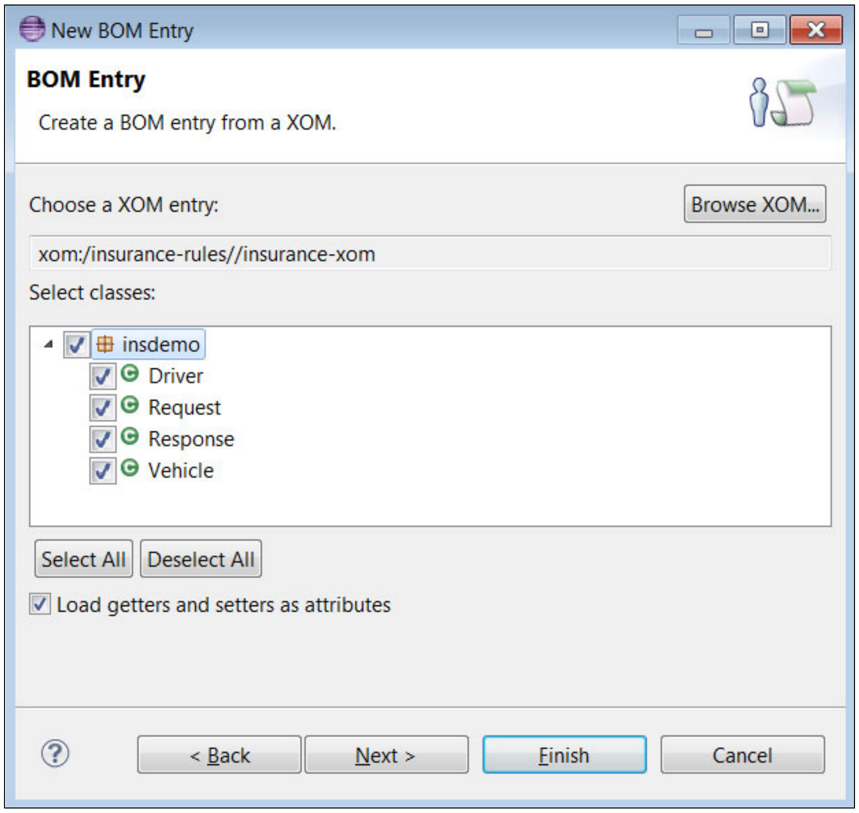


Figure 3-18   Selecting a generated Java XOM

4. In the Select classes field of the BOM Entry dialog, select the **XOM** package. When you select the package, you automatically select all the classes that it contains, as shown in Figure 3-19. Click **Finish**.



**BOM Entry**

Create a BOM entry from a XOM.

Choose a XOM entry:

xom:/insurance-rules//insurance-xom

Select classes:

- [✓] insdemo
  - [✓] Driver
  - [✓] Request
  - [✓] Response
  - [✓] Vehicle

[✓] Load getters and setters as attributes

Figure 3-19 Selecting a XOM package to import all classes

5. In the Rule Explorer view, the `bom` folder contains a new BOM entry model, as shown in Figure 3-20.

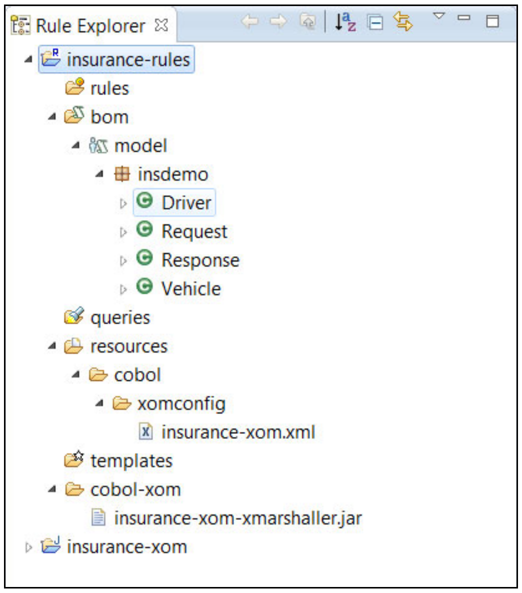


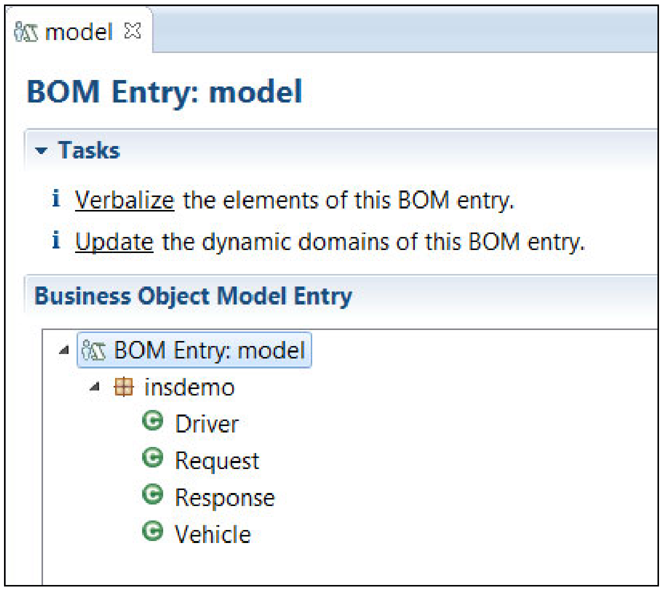
Figure 3-20 Viewing the generated BOM

6. View the generated BOM and its verbalization:

   a. In the Rule Explorer view, Expand **bom** then double-click **model** to open the BOM editor.

   b. In the BOM editor, expand the insdemo package to view the generated BOM, as shown in Figure 3-21.

   c. Double-click the **Driver** class to view the default class verbalization.



model

BOM Entry: model

Tasks

Verbalize the elements of this BOM entry.

Update the dynamic domains of this BOM entry.

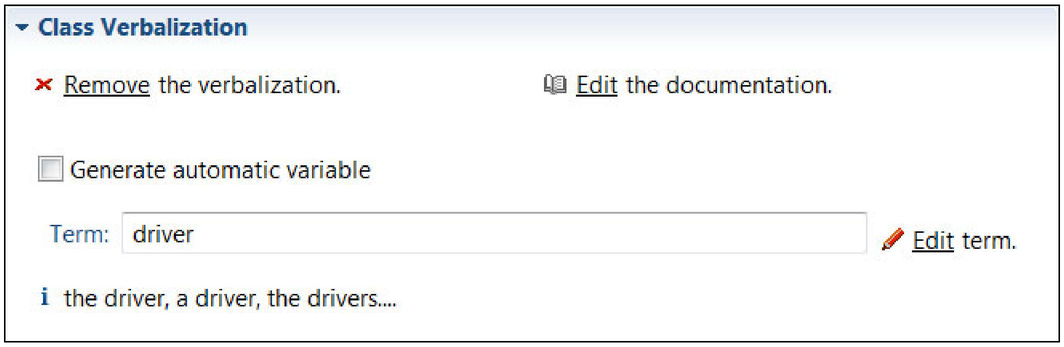
Business Object Model Entry

- BOM Entry: model
  - insdemo
    - Driver
    - Request
    - Response
    - Vehicle

*Figure 3-21 Viewing the generated BOM Entry model in the BOM editor*

   d. The resulting Class Verbalization section (of the Class Driver window) is shown (Figure 3-22).

**Language:** The default verbalization is in English. If you are working in a localized version of Rule Designer, you can verbalize the BOM classes in the language of your locale.



Class Verbalization

Remove the verbalization.

Edit the documentation.

Generate automatic variable

Term: driver

Edit term.

the driver, a driver, the drivers....

*Figure 3-22 Viewing the default verbalization*

## 3.2.5 Declaring ruleset parameters

*Ruleset parameters* provide the means to exchange data between a COBOL application and the rule application. You define ruleset parameters by name, type, and direction.

In this example, you decide on the status of an insurance request and response, so that you create ruleset parameters for the Request and Response classes. You use the IN direction for the request parameter. The value of the request parameter is provided as input from the COBOL client application on execution. The direction for the response parameter must be IN_OUT. The value of the request parameter is set by the IN value passed by the client and then updated by the engine on the way OUT. The updated value is returned to the client.

**Important:** You *cannot* use the OUT parameter direction with zRES, because COBOL programs do not support memory allocation dynamically.

Follow these steps to declare ruleset parameters:

1. In the Design part of the Rule Project Map tab, click **Define parameters**, as shown in Figure 3-23.

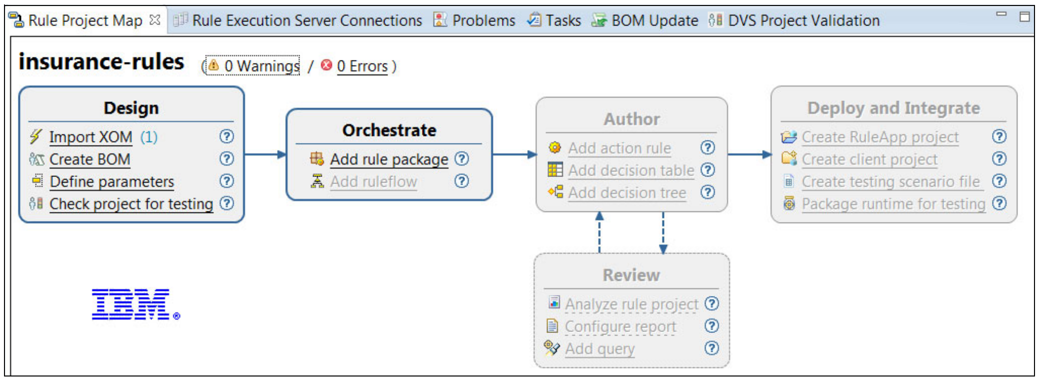
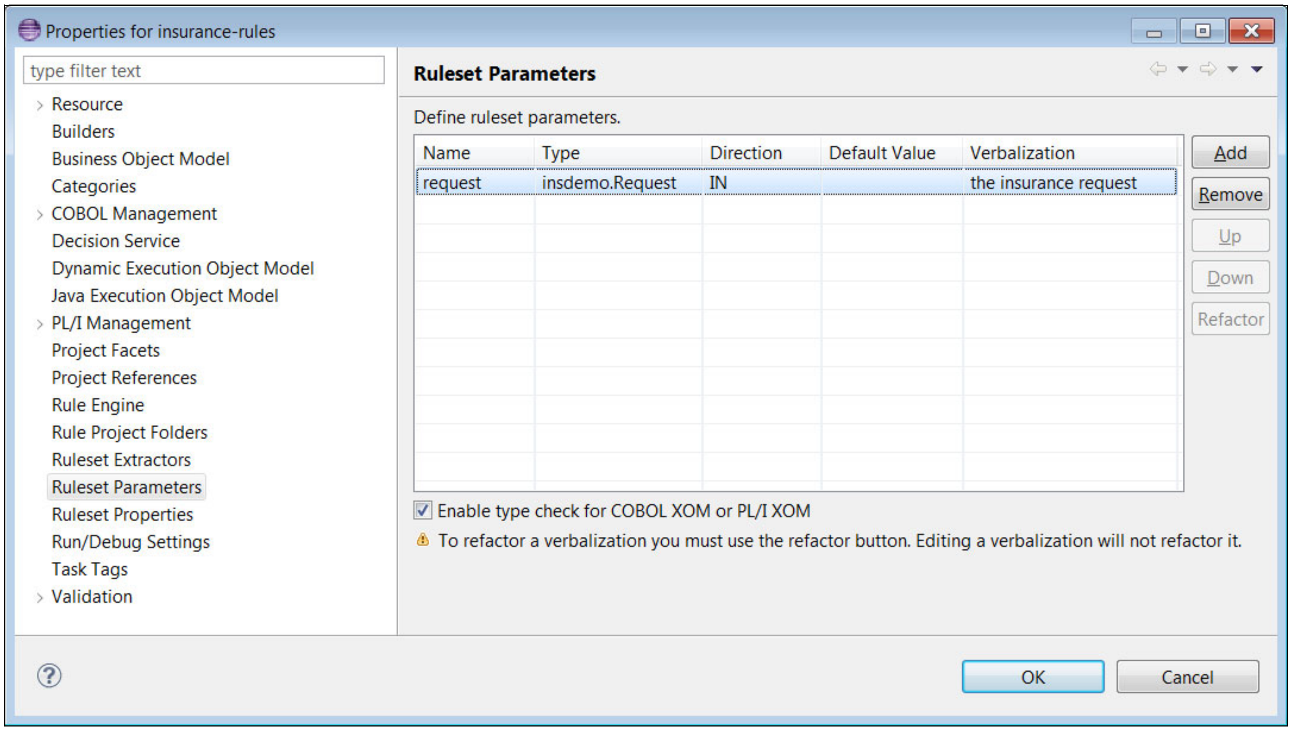


*Figure 3-23 Selecting Define parameters option*

2. In the Ruleset Parameters dialog, select **Enable type check for COBOL XOM**.

3. To define a request parameter, click **Add**. Then, change the following default values, as shown in Figure 3-24:
   - In the Name column, delete `myParam` and type `request`.
   - In the Type column, click the ellipsis (**...**) on the right of the cell, and select **Request** (Figure 3-24). The `insdemo.Request` entry is entered in the cell automatically.
   - In the Direction column, select the **IN** direction.
   - In the Verbalization column, type `the insurance request`.



**Ruleset Parameters**

Define ruleset parameters.

| Name | Type | Direction | Default Value | Verbalization |
|---|---|---|---|---|
| request | insdemo.Request | IN | | the insurance request |

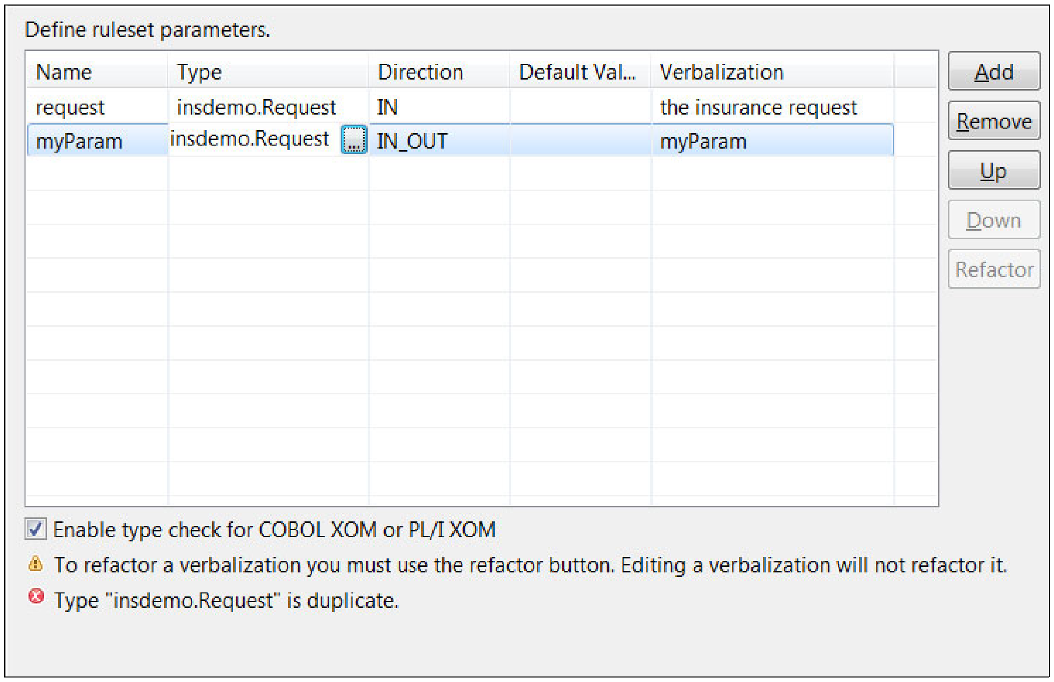
✓ Enable type check for COBOL XOM or PL/I XOM

To refactor a verbalization you must use the refactor button. Editing a verbalization will not refactor it.

*Figure 3-24 Adding the ruleset parameter for request*

4. To define the response parameter, click **Add**. Then, change the following default values, as shown in Figure 3-25. A completed example is shown in Figure 3-27 on page 42:
   - In the Name column, delete `myParam` and type `response`.
   - In the Type column, the `insdemo.Request` entry is entered in the cell automatically, click the ellipsis (**...**) on the right of the cell (Figure 3-25), and select **Response** (Figure 3-26 on page 42). The `insdemo.Response` entry is entered in the cell automatically.
   - In the Direction column, select the **IN_OUT** direction.
   - In the Verbalization column, type `the insurance response`.

   Click **OK**.


Define ruleset parameters.

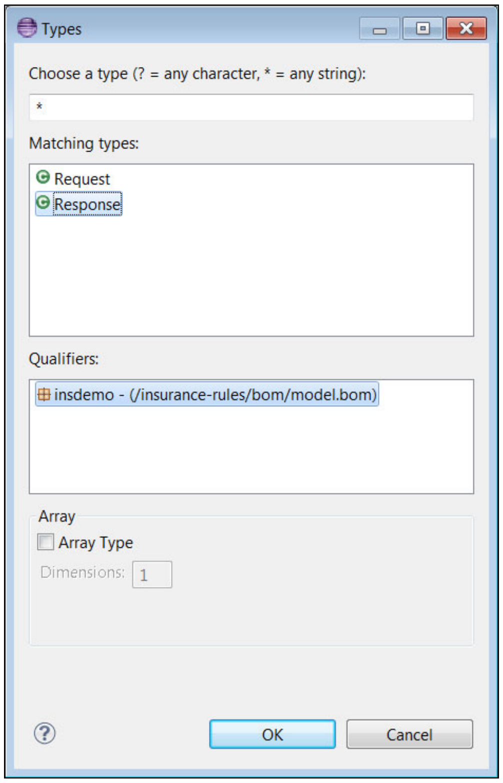
| Name | Type | Direction | Default Val... | Verbalization |
|---|---|---|---|---|
| request | insdemo.Request | IN | | the insurance request |
| myParam | insdemo.Request | IN_OUT | | myParam |

☑ Enable type check for COBOL XOM or PL/I XOM

To refactor a verbalization you must use the refactor button. Editing a verbalization will not refactor it.

Type "insdemo.Request" is duplicate.

*Figure 3-25 Default Response ruleset parameters*


Types

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

*

Matching types:

- Request
- Response

Qualifiers:

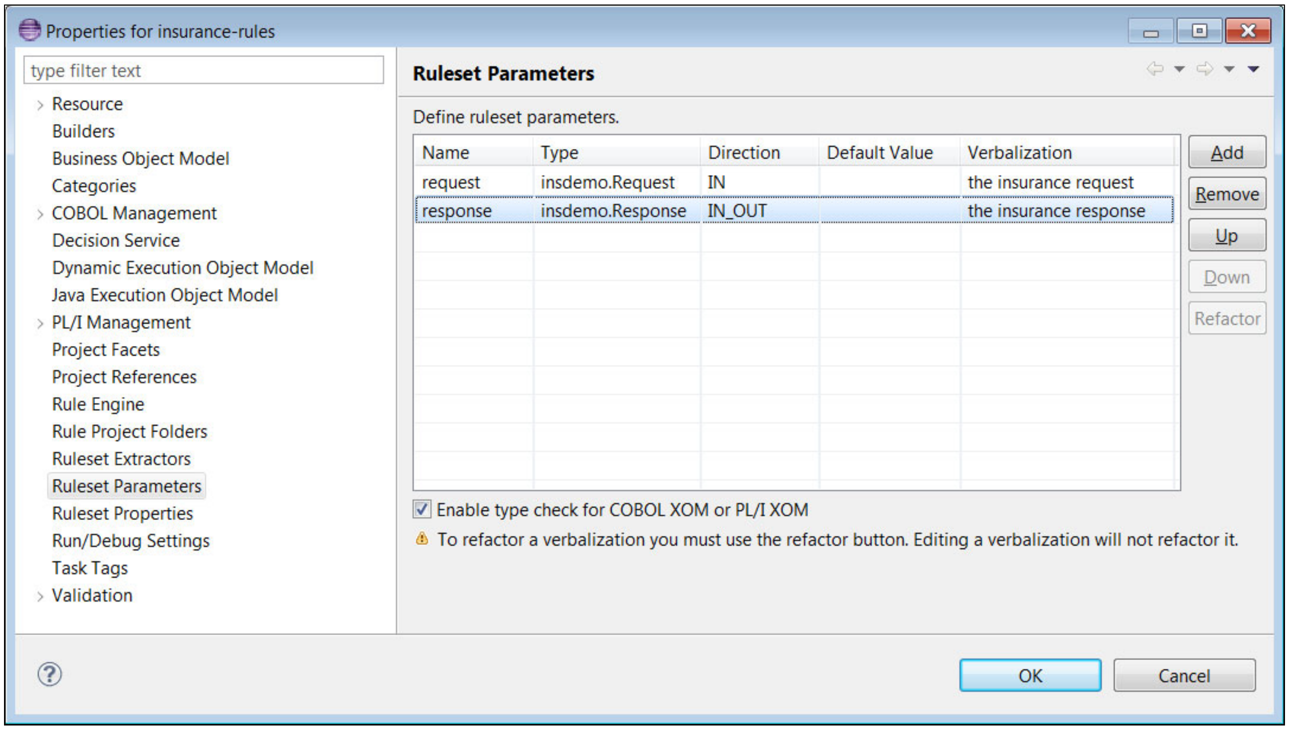
- insdemo - (/insurance-rules/bom/model.bom)

Array

- [ ] Array Type

Dimensions: 1

*Figure 3-26 Selecting ruleset parameter type*


Properties for insurance-rules

type filter text

- Resource
- Builders
- Business Object Model
- Categories
- COBOL Management
- Decision Service
- Dynamic Execution Object Model
- Java Execution Object Model
- PL/I Management
- Project Facets
- Project References
- Rule Engine
- Rule Project Folders
- Ruleset Extractors
- Ruleset Parameters
- Ruleset Properties
- Run/Debug Settings
- Task Tags
- Validation

**Ruleset Parameters**

Define ruleset parameters.

| Name | Type | Direction | Default Value | Verbalization |
|---|---|---|---|---|
| request | insdemo.Request | IN | | the insurance request |
| response | insdemo.Response | IN_OUT | | the insurance response |

- [✓] Enable type check for COBOL XOM or PL/I XOM

To refactor a verbalization you must use the refactor button. Editing a verbalization will not refactor it.

*Figure 3-27 Add the ruleset parameter for response*

## 3.2.6 Adding BOM methods and mapping them to the XOM

You use methods to specify conditions and actions in your rules. You create methods in the Rule Designer. When you add methods to the BOM, you use BOM-to-XOM mapping in the BOM editor to implement the method.

**Important:** You *cannot* map the BOM method to a Java XOM method, because you must not change the XOM.

This section describes how to add the following BOM methods:

- addMessage: Defines what is needed to pass information from the rules.
- reject: Identifies whether the insurance request was rejected.

### Adding the addMessage method

To add the addMessage method, follow these steps:

1. In the Rule Explorer view, expand the model package, and double-click the **Response** class, as shown in Figure 3-28.

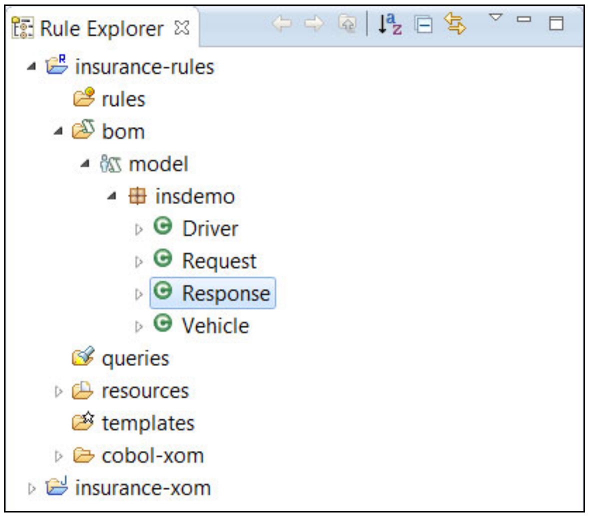


*Figure 3-28 Selecting the Response class*

2. On the Class Response page of the BOM editor, to the right of the Members section, click **New**, as shown in Figure 3-29.

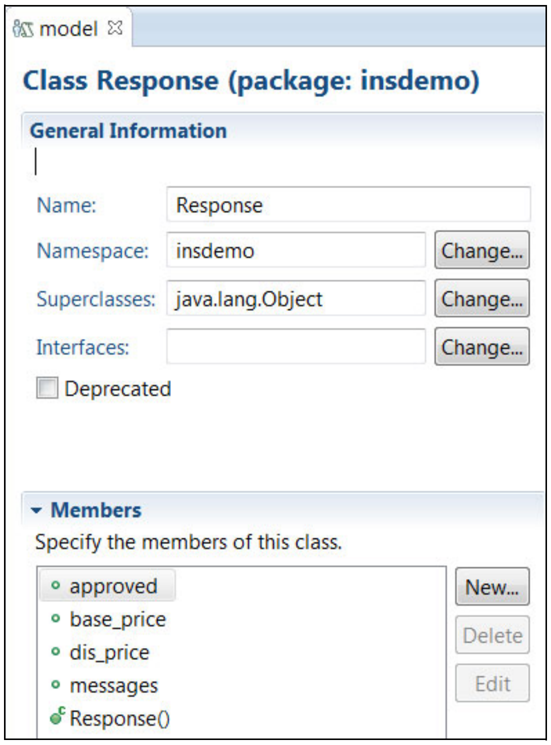


Figure 3-29   Creating a member

3. In the New Member dialog (Figure 3-30), enter the following information:

   – For the Type, select **Method**.
   – For the Name, enter `addMessage`.
   – For the Type, enter `void`.

   Click **Add**.

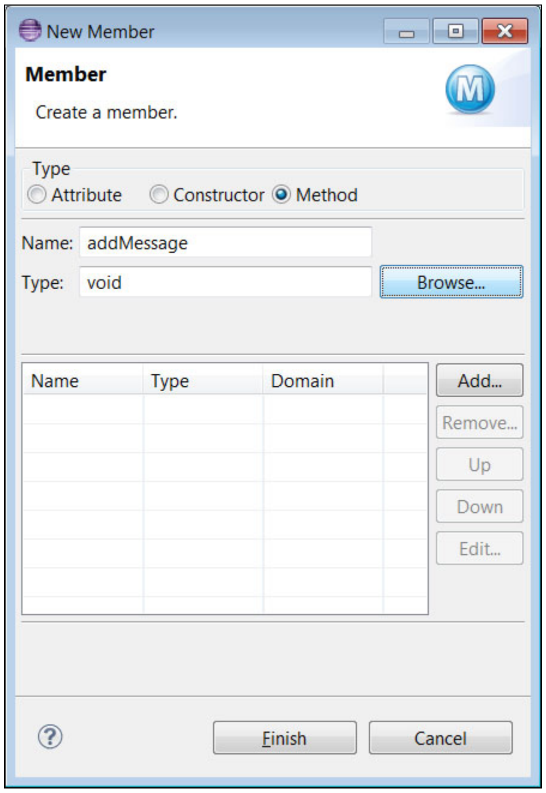


*Figure 3-30 Creating a method for addMessage*

4. In the Method Argument dialog (Figure 3-31), enter the following information:

   – For the Name, enter `msg`.
   – For the Type, enter `java.lang.String`.

   Click **OK**, and then click **Finish** on the New Member dialog.

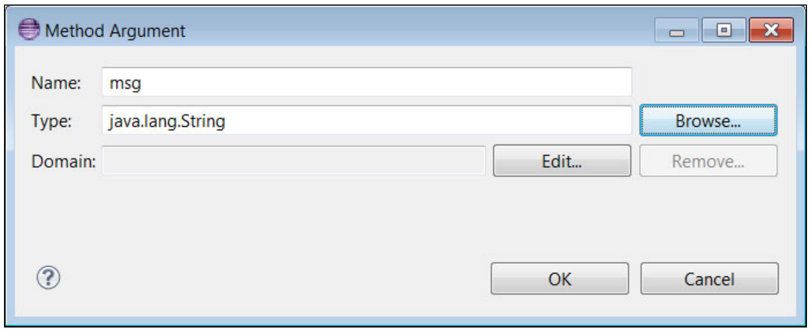


*Figure 3-31 Adding the method argument*

5. On the Class page of the BOM editor, the Members list now includes the addMessage(String) method, as shown in Figure 3-32. Double-click the **addMessage** method.

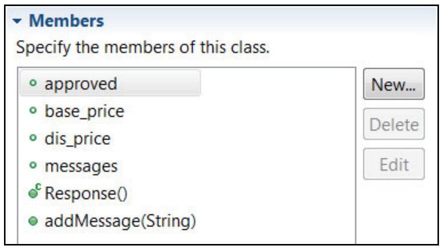
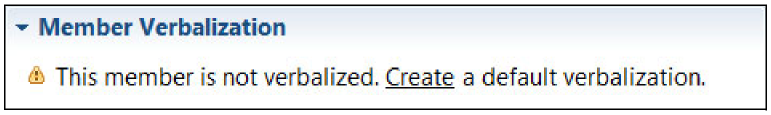


*Figure 3-32 addMessage method created*

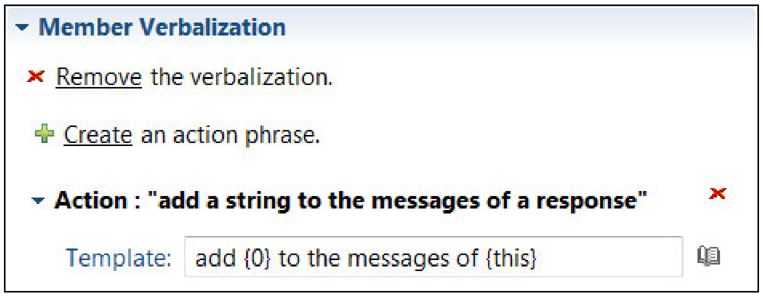
6. In the Member Verbalization section of the BOM editor (Figure 3-33), click **Create** to view the default verbalization.


Member Verbalization

This member is not verbalized. Create a default verbalization.

*Figure 3-33 Creating verbalization*

7. The default verbalization of the addMessage class is now displayed. Keep the default verbalization of **add {0} to the messages of {this}**, as shown in Figure 3-34.


Member Verbalization

Remove the verbalization.

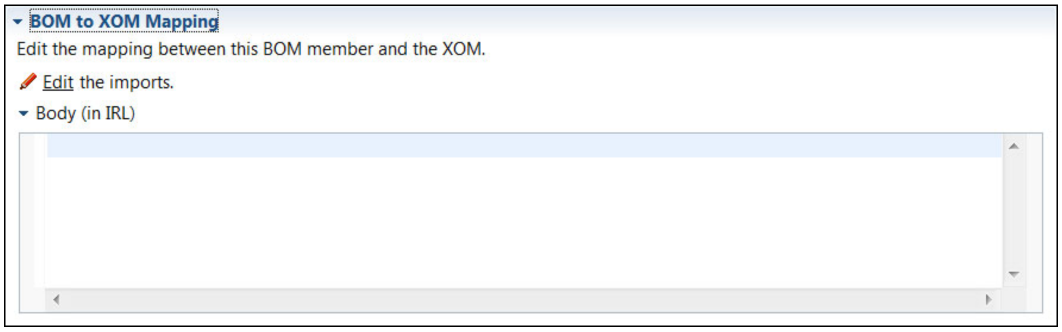
Create an action phrase.

Action : "add a string to the messages of a response"

Template: add {0} to the messages of {this}

*Figure 3-34 Keeping the default verbalization*

8. Scroll down to the BOM-to-XOM mapping section of the BOM editor and expand it to activate the BOM-to-XOM mapping editor, as shown in Figure 3-35.


BOM to XOM Mapping

Edit the mapping between this BOM member and the XOM.

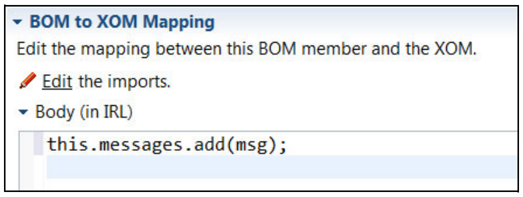
Edit the imports.

Body (in IRL)

*Figure 3-35 Activating the BOM-to-XOM mapping editor*

9. Enter the following Java code (Figure 3-36):

```
this.messages.add(msg);
```



```
this.messages.add(msg);
```

*Figure 3-36 Adding method implementation*

10. Save your work.

## Adding the reject method

To add the reject method, follow these steps:

1. Double-click the **Response** class in the Rule Explorer. Under the Members section, click **New**, as shown in Figure 3-37.

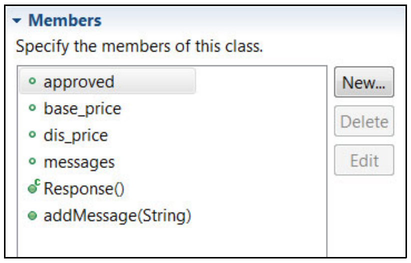
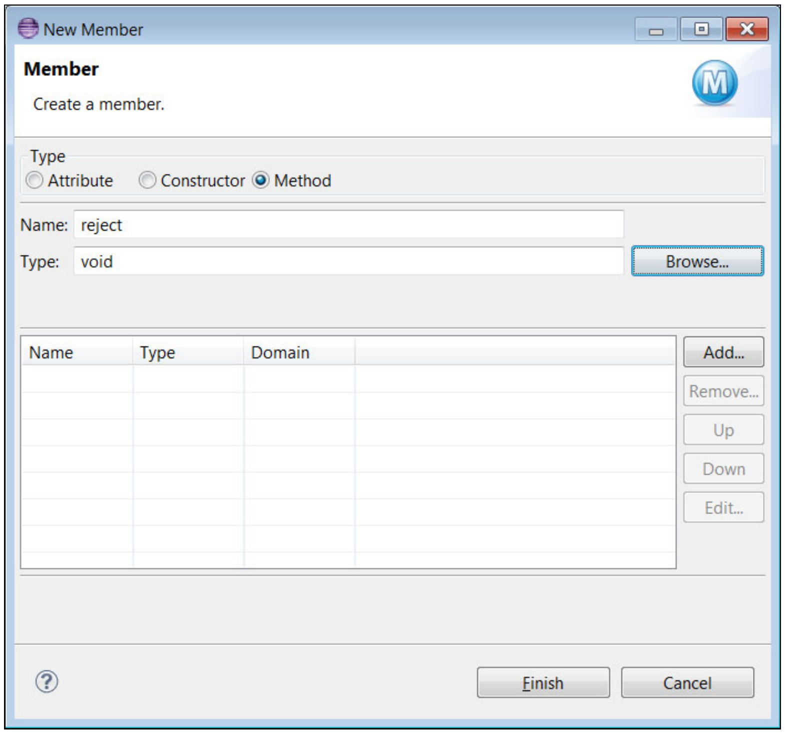


*Figure 3-37 Creating a new reject method*

2. In the New Member dialog (Figure 3-38), enter the following information:
   – For the Type, select **Method**.
   – For the Name, enter `reject`.
   – For the Type, enter `void`.

   Click **Finish**.



New Member

Member

Create a member.

Type

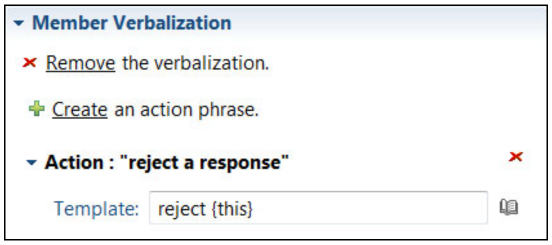
Attribute [✓ Method] Constructor

Name: reject

Type: void

*Figure 3-38 Defining the method argument*

3. Create the default verbalization for the reject method. Double-click the `reject()` method. Then, click **Create** and accept the default verbalization of `reject {this}`, as shown in Figure 3-39.



Member Verbalization

Remove the verbalization.

Create an action phrase.

Action : "reject a response"

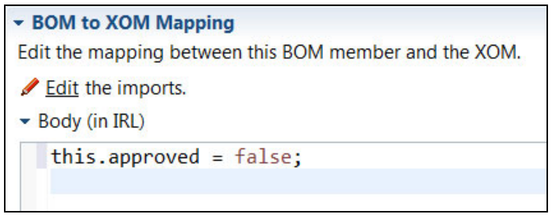
Template: reject {this}

*Figure 3-39 Defining verbalization for the reject method*

4. Scroll down to the BOM-to-XOM mapping section of the BOM editor and expand it to activate the BOM-to-XOM mapping editor. Enter the following Java code, as shown in Figure 3-40:

   Type `this.approved = false;`

   Save your work.


BOM to XOM Mapping

Edit the mapping between this BOM member and the XOM.

Edit the imports.

Body (in IRL)

```
this.approved = false;
```

*Figure 3-40 Implementing the reject method*

The Rule Explorer now shows that these members are present in their classes, as shown in Figure 3-41.

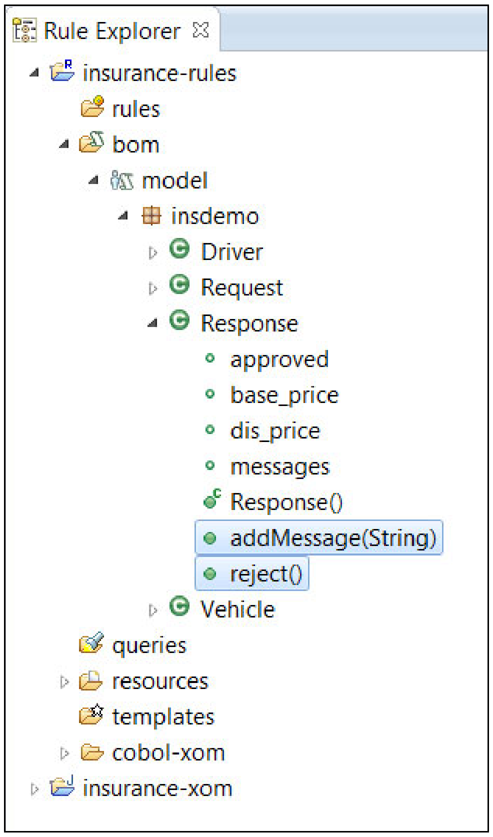

```
Rule Explorer
insurance-rules
    rules
    bom
        model
            insdemo
                Driver
                Request
                Response
                    approved
                    base_price
                    dis_price
                    messages
                    Response()
                    addMessage(String)
                    reject()
                Vehicle
    queries
    resources
    templates
    cobol-xom
insurance-xom
```

*Figure 3-41 Viewing the new BOM methods*

## 3.2.7 Creating the ruleflow

Before writing the rules, you orchestrate how the rules execute. You control the order in which rules are executed by using *ruleflows*. When defining the flow of execution, you organize rules into packages that contain related rules. This section explains how to create a package that relates to the validation rules.

Follow these steps to create a ruleflow:

1. In Rule Designer, in the IBM Orchestrate® part of the Rule Project Map, click **Add rule package**, (Figure 3-42).

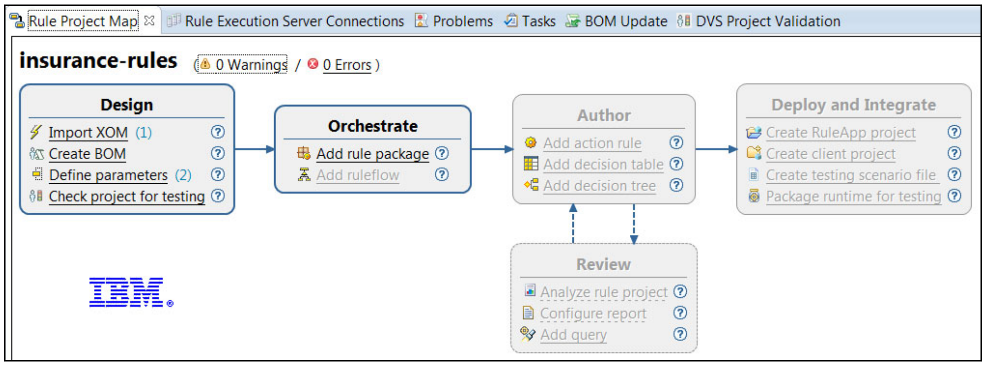
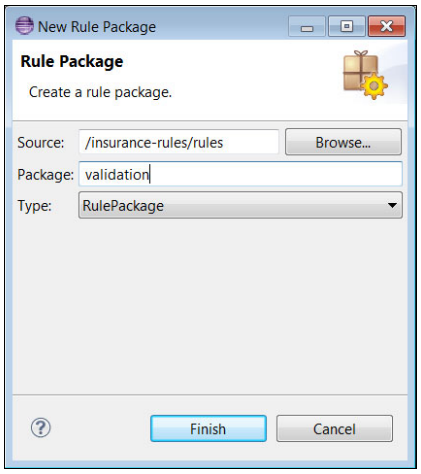


Figure 3-42 Adding a rule package

2. In the New Rule Package dialog, in the Package field, enter `validation` and click **Finish**, (Figure 3-43).


New Rule Package

**Rule Package**

Create a rule package.

Source: /insurance-rules/rules

Package: validation

Type: RulePackage

Figure 3-43 Entering the validation package name

3. To create the ruleflow, in the Orchestrate part of the Rule Project Map, click **Add ruleflow**, (Figure 3-44).

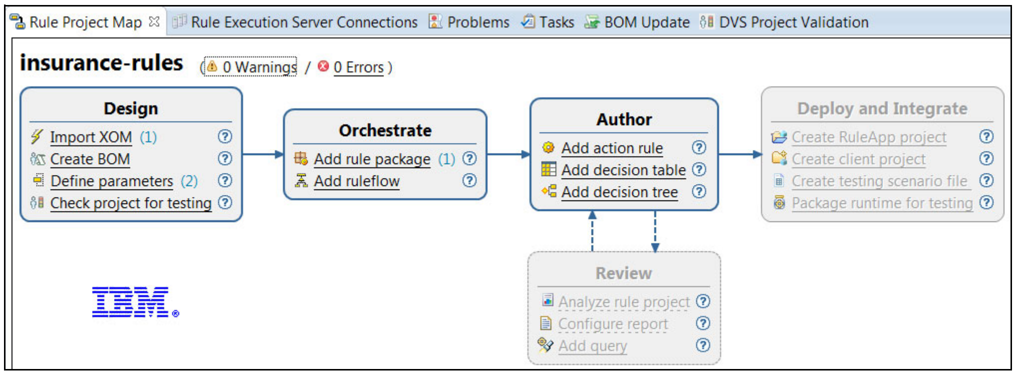
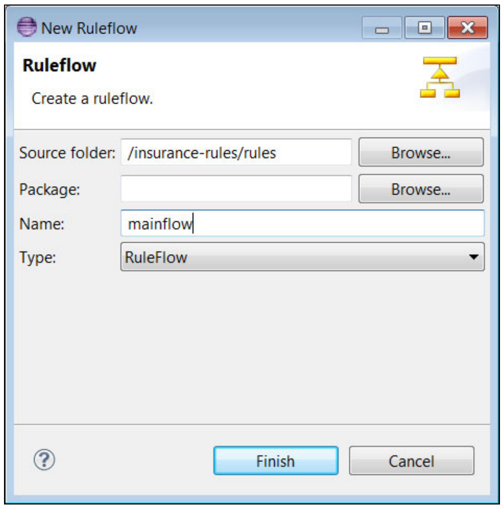


*Figure 3-44 Adding a new ruleflow*

4. In the New Ruleflow dialog, in the Name field, enter `mainflow` and click **Finish** (Figure 3-45).



Source folder: /insurance-rules/rules
Package:
Name: mainflow
Type: RuleFlow

*Figure 3-45 Entering the ruleflow name*

5. To create a ruleflow by using the Ruleflow diagram that is shown in Figure 3-46 and Figure 3-47, complete the following steps:

   a. Add the start node, which is the starting point of the ruleflow. Click the start node icon ( ) in the ruleflow diagram toolbar and drop it in the ruleflow diagram.

   b. Add the end node, which is the endpoint of the ruleflow. Click the end node icon ( ) in the ruleflow diagram toolbar and drop it in the ruleflow diagram.

   c. Add the task for the validation rule package, which is the rule task of the ruleflow. Click the **validation** rule package in the Rule Explorer view and drag it into the ruleflow diagram, as shown in Figure 3-46.

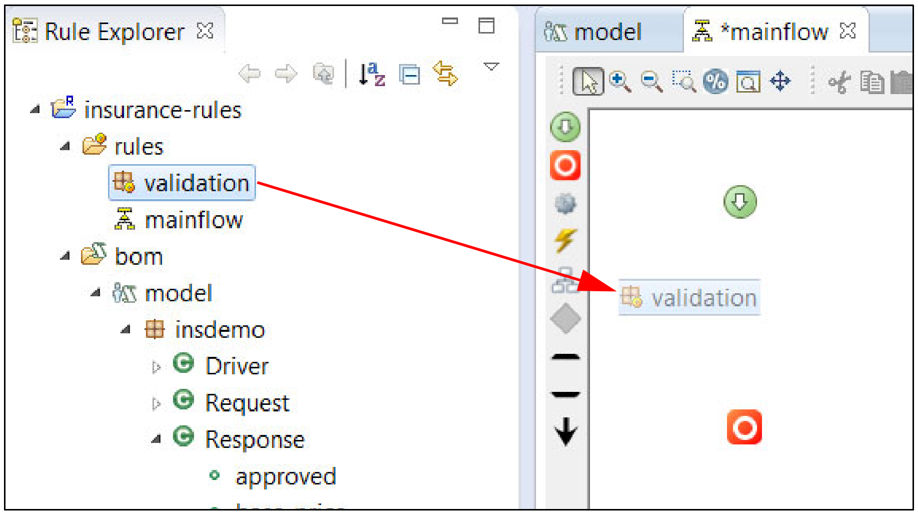


*Figure 3-46 Dragging the validation package to the canvas*

   d. Connect the elements (as shown in Figure 3-47):
      i. Click the arrow icon ( ) to start connection mode.
      ii. Click the start node icon ( ) and then click the **validation** task box.
      iii. Click the **validation** task box again, and finally, click the end node icon ( ).

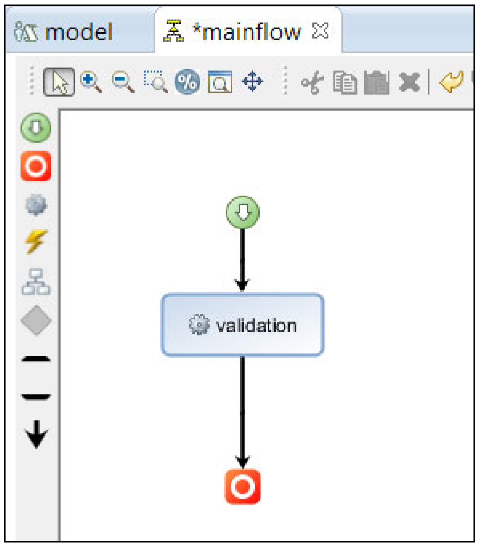


*Figure 3-47 Designing the ruleflow*

6. Next, you can optionally refine the diagram by clicking the icon. Figure 3-48 shows the diagram.

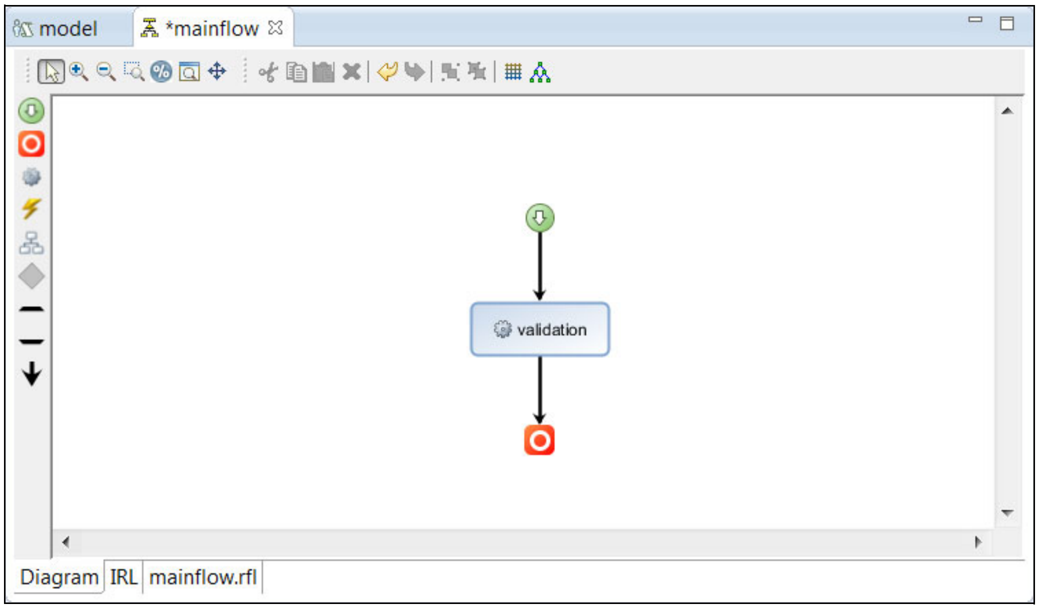


*Figure 3-48 Refining the ruleflow*

7. Save your work.

## 3.2.8 Authoring rules

This section explains how to write action rules and put them into the relevant package. You can create the following rules in Rule Designer for the validation packages:

- MaxiMinimumAge rule
- NumberOfAccidents rule

To create the action rules, follow these steps:

1. Create the MaxiMinimumAge rule:

   a. In the rules project, right-click "validation" and then click **New** → **Action Rule**, as indicated in Figure 3-49.

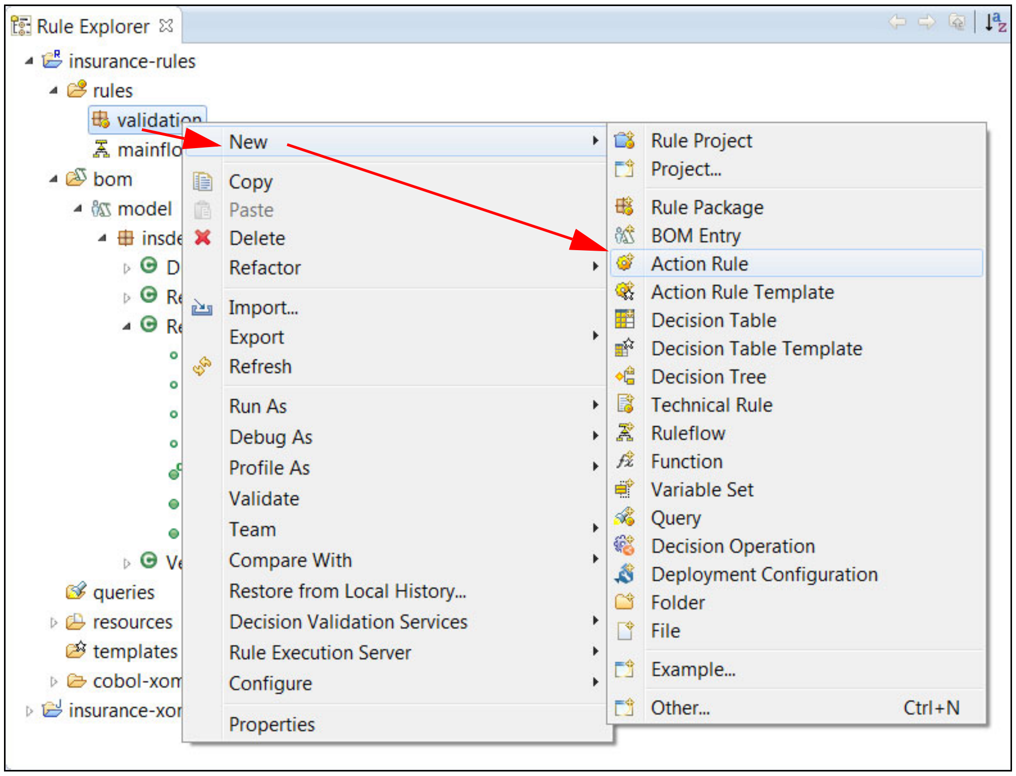
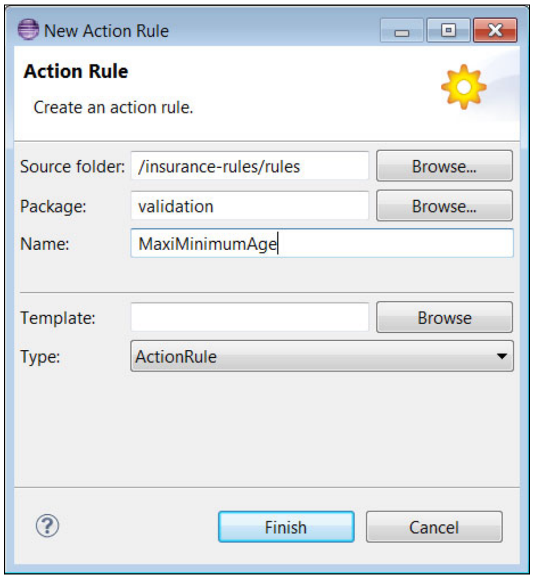


*Figure 3-49 Creating an action rule*

   b. In the New Action Rule dialog, enter `MaxiMinimumAge` in the Name field, as shown in Figure 3-50. Click **Finish**.



Source folder: /insurance-rules/rules
Package: validation
Name: MaxiMinimumAge
Template:
Type: ActionRule

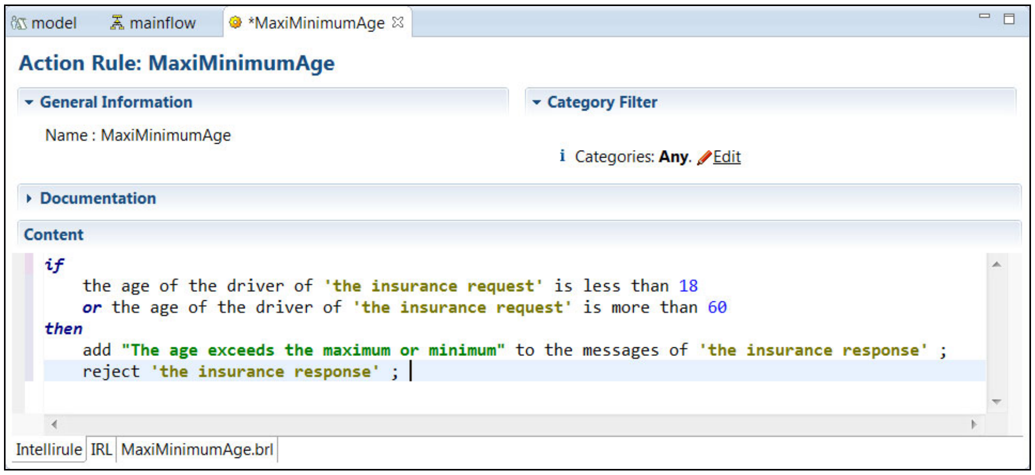
*Figure 3-50 Entering the rule name*

c. The new action rule, MaxiMinimumAge, is displayed in the Rule Explorer view and the Intellirule editor opens. Enter the validation with the MaxiMinimumAge rule, as shown in Example 3-2.

*Example 3-2 Validation with the MaxiMinimumAge rule*

```
if
    the age of the driver of 'the insurance request' is less than 18
    or the age of the driver of 'the insurance request' is more than 60
then
    add "The age exceeds the maximum or minimum" to the messages of 'the
insurance response' ;
    reject 'the insurance response' ;
```

Figure 3-51 shows the generated action rule.



Name : MaxiMinimumAge

Categories: Any. Edit

```
if
    the age of the driver of 'the insurance request' is less than 18
    or the age of the driver of 'the insurance request' is more than 60
then
    add "The age exceeds the maximum or minimum" to the messages of 'the insurance response' ;
    reject 'the insurance response' ;
```

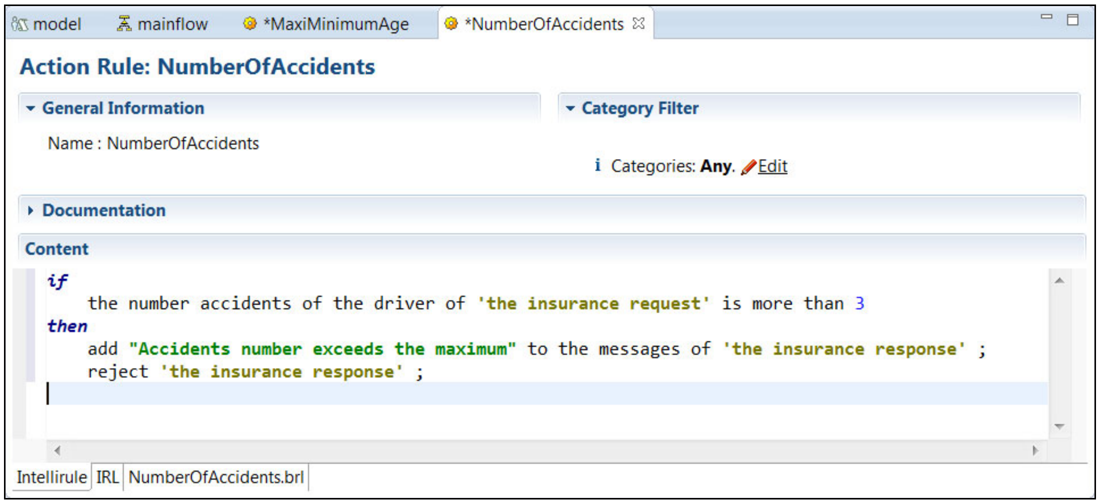
*Figure 3-51 Viewing the MaxiMinimumAge rule*

2. Create a second action rule by repeating step 1 on page 54 with a name of `NumberOfAccidents`, as shown in Example 3-3.

*Example 3-3 Validation with the NumberOfAccidents rule*

```
if
    the number accidents of the driver of 'the insurance request' is more than
3
then
    add "Accidents number exceeds the maximum" to the messages of 'the
insurance response' ;
    reject 'the insurance response' ;
```

Figure 3-52 shows the generated action rule.


model | mainflow | *MaxiMinimumAge | *NumberOfAccidents

Action Rule: NumberOfAccidents

General Information

Name : NumberOfAccidents

Category Filter

Categories: **Any**. Edit

Documentation

Content

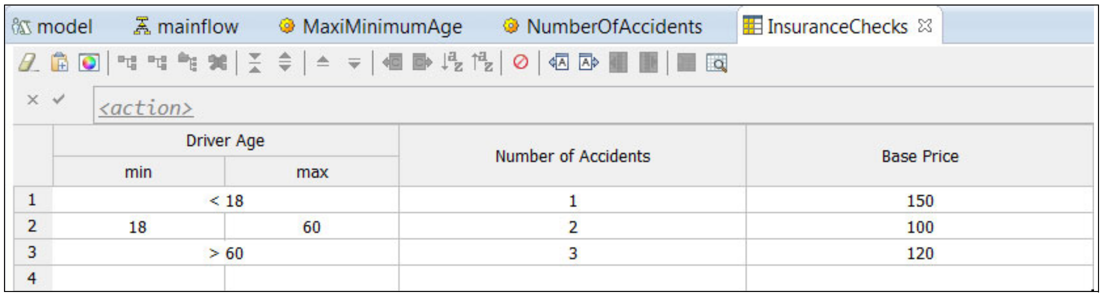
```
if
    the number accidents of the driver of 'the insurance request' is more than 3
then
    add "Accidents number exceeds the maximum" to the messages of 'the insurance response' ;
    reject 'the insurance response' ;
```

Intellirule | IRL | NumberOfAccidents.brl

*Figure 3-52 View the generated rules*

3. Save your work.

You can also use a decision table or decision tree to write decision rules, as shown in the example in Figure 3-53.


model | mainflow | MaxiMinimumAge | NumberOfAccidents | InsuranceChecks

<action>

| | Driver Age | | Number of Accidents | Base Price |
|---|---|---|---|---|
| | min | max | | |
| 1 | < 18 | | 1 | 150 |
| 2 | 18 | 60 | 2 | 100 |
| 3 | > 60 | | 3 | 120 |
| 4 | | | | |

*Figure 3-53 Decision table example*

## 3.2.9 Preparing the rule execution

This section shows you how to deploy rules to the zRule Execution Server for z/OS and how to view the deployed ruleset on the zRule Execution Server for z/OS web console.

## Step 1: Creating a RuleApp project

First, you must create a RuleApp project to contain the rulesets that you want to execute. To create a RuleApp project, follow these steps:

1. In Rule Designer, in the Deploy and Integrate section of the Rule Project Map, click **Create RuleApp project** (Figure 3-54).

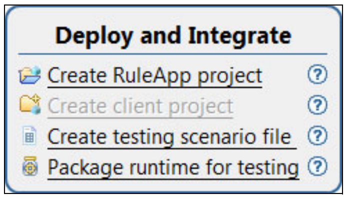
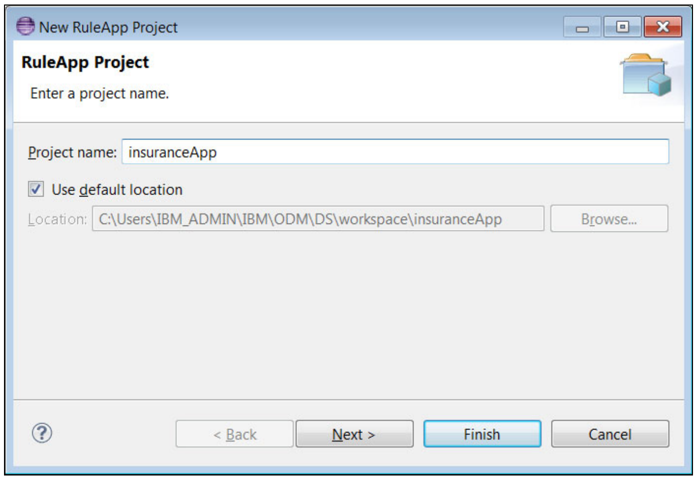


*Figure 3-54 Creating a RuleApp project*

2. In the New RuleApp Project dialog, enter `insuranceApp` in the Project name field as the name for your RuleApp project. Ensure that **Use default location** is selected (Figure 3-55). Click **Next**.



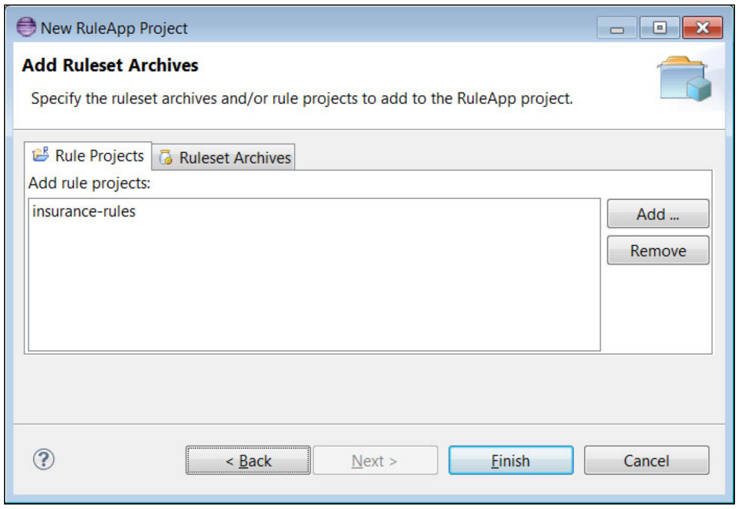
Project name: insuranceApp

☑ Use default location

Location: C:\Users\IBM_ADMIN\IBM\ODM\DS\workspace\insuranceApp

*Figure 3-55 Entering the RuleApp name*

3. The rule project is listed in the Rule Projects tab, as shown in Figure 3-56. Click **Finish**.



Add Ruleset Archives

Specify the ruleset archives and/or rule projects to add to the RuleApp project.

Add rule projects:

insurance-rules

*Figure 3-56 Viewing the Rule Projects tab*

4. The RuleApp project is created and displayed in the Rule Explorer view, as shown in Figure 3-57.

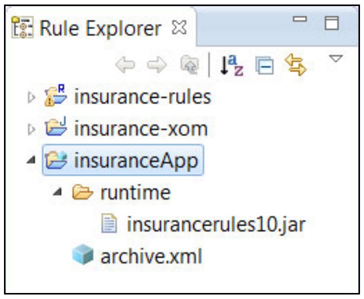


*Figure 3-57 Viewing the insuranceApp project*

## Step 2: Deploying the RuleApp to the zRule Execution Server for z/OS

To be able to execute the ruleset with zRule Execution Server for z/OS, you must deploy the following artifacts to zRule Execution Server for z/OS:

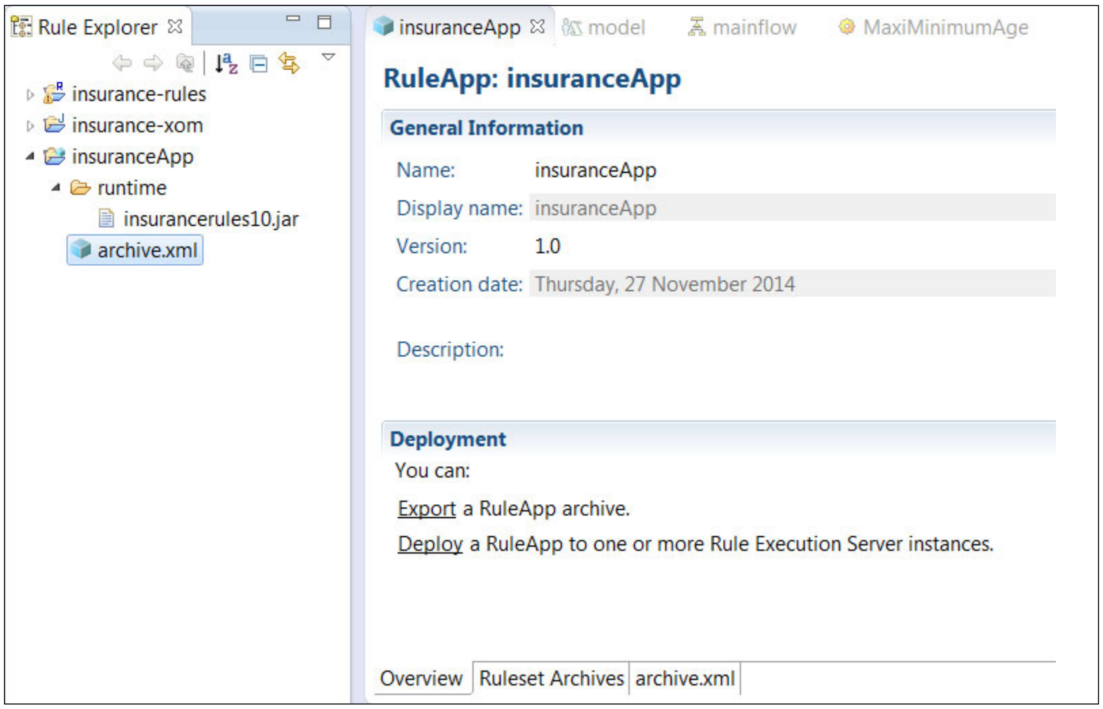
- RuleApps containing the business rules within rulesets
- A JAR resource or library that contains Java classes that are used by the rules

**Important:** The RuleApp must be deployed to zRule Execution Server for z/OS. Ensure that you start zRule Execution Server for z/OS successfully before you continue this step. For information, see the *Deploying RuleApps and XOMs* topic in the IBM Operational Decision Manager Version 8.7.1 IBM Knowledge Center:

`http://www-01.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.zres/topics/con_zres_deploy_ruleapps_xoms.html`

To deploy the XOM, marshaller, and RuleApp, follow these steps:

1. Within the insuranceApp RuleApp project, double-click the `archive.xml` file to open the RuleApp editor (Figure 3-58).



RuleApp: insuranceApp

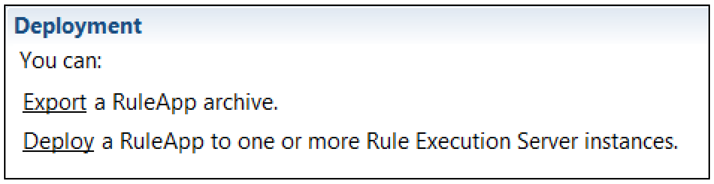
General Information

Name: insuranceApp
Display name: insuranceApp
Version: 1.0
Creation date: Thursday, 27 November 2014
Description:

Deployment
You can:
Export a RuleApp archive.
Deploy a RuleApp to one or more Rule Execution Server instances.

*Figure 3-58 RuleApp editor for insuranceApp*

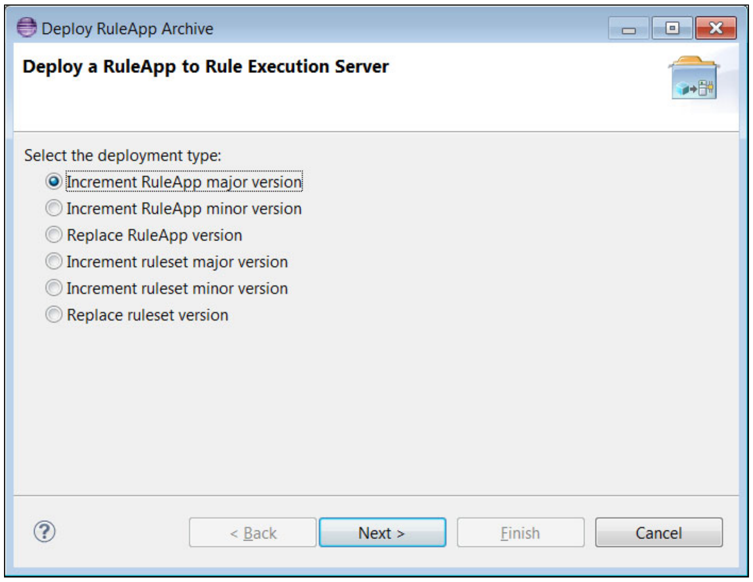
2. In the Deployment pane, click **Deploy** to deploy the RuleApp to the Rule Execution Server (Figure 3-59).



Deployment
You can:
Export a RuleApp archive.
Deploy a RuleApp to one or more Rule Execution Server instances.

*Figure 3-59 Deploying the RuleApp*

3. In the Deploy RuleApp Archive dialog, select the default option **Increment RuleApp major version** for deployment type (Figure 3-60) and click **Next**.



Deploy RuleApp Archive

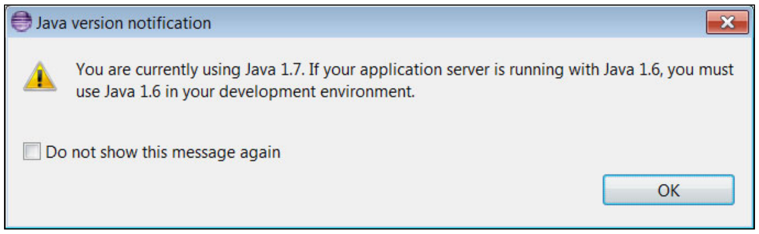
Deploy a RuleApp to Rule Execution Server

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

*Figure 3-60 Selecting the deployment type*

4. If your system is running Java 1.7, you receive a java version notification warning window (Figure 3-61). Effective with Version 8.5, the default Java environment for the IBM Operational Decision Manager, IBM Operational Decision Manager for z/OS, and IBM Business Rules for z/OS Rule Designers became Java 7. While this is satisfactory for a wide variety of decision management execution environments, there are situations where a Java 6 environment is required. One example is using the zRule Execution Server for z/OS in a CICS V4.2 environment. Both Java 6 and Java 7 are supported environments on all platforms for the Operational Decision Manager and Business Rules rule products.



Java version notification

You are currently using Java 1.7. If your application server is running with Java 1.6, you must use Java 1.6 in your development environment.

Do not show this message again

*Figure 3-61 Java version notification warning*

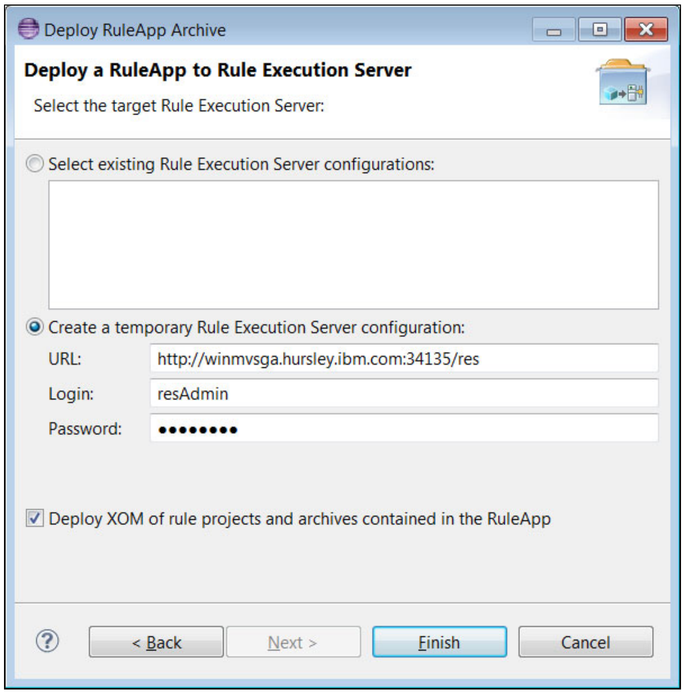
**Note:** A document detailing how to set up a Java 6 compliant environment for the Rule Designer can be found at the following location:

`http://www-01.ibm.com/support/docview.wss?uid=swg216914941`

5. Select **Create a temporary Rule Execution Server configuration** and enter the following details, as shown in Figure 3-62:

   – URL: `http://<your.server.address>:<PORT>/res`
   – Login: `resAdmin`
   – Password: `resAdmin`

   Select **Deploy XOM of rule projects and archives contained in the RuleApp** and click **Finish**.


Deploy RuleApp Archive
Deploy a RuleApp to Rule Execution Server
Select the target Rule Execution Server:
Select existing Rule Execution Server configurations:
< Back
Next >
Finish
Cancel

Create a temporary Rule Execution Server configuration: (selected)

URL: http://winmvsga.hursley.ibm.com:34135/res

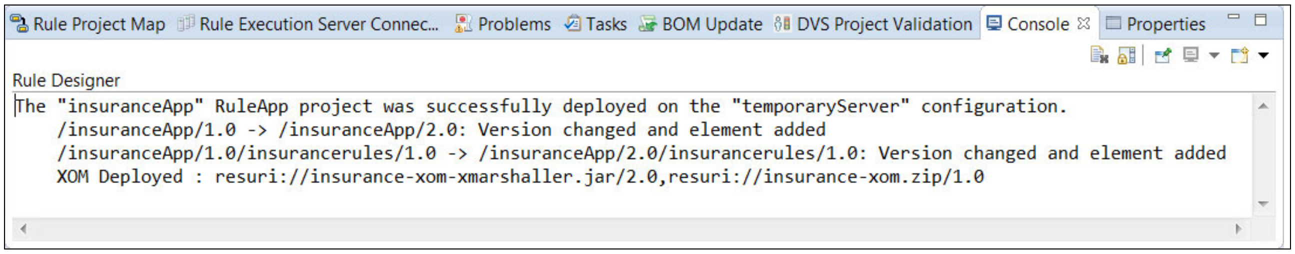
Login: resAdmin

Password: ••••••••

✓ Deploy XOM of rule projects and archives contained in the RuleApp

*Figure 3-62 Configuring the RuleApp deployment*

In the Console tab, you can see the confirmation that the project has been deployed, as shown in Figure 3-63. The artifacts are now deployed to the zRule Execution Server for z/OS server.



```
The "insuranceApp" RuleApp project was successfully deployed on the "temporaryServer" configuration.
    /insuranceApp/1.0 -> /insuranceApp/2.0: Version changed and element added
    /insuranceApp/1.0/insurancerules/1.0 -> /insuranceApp/2.0/insurancerules/1.0: Version changed and element added
    XOM Deployed : resuri://insurance-xom-xmarshaller.jar/2.0,resuri://insurance-xom.zip/1.0
```

*Figure 3-63 Deploying the RuleApp confirmation*

## Step 3: Viewing deployed rule artifacts in the Rule Execution Server console

You can log in to the Rule Execution Server console and use the Navigator pane to view the deployed RuleApp and XOM. To view your deployed artifacts, follow these steps:

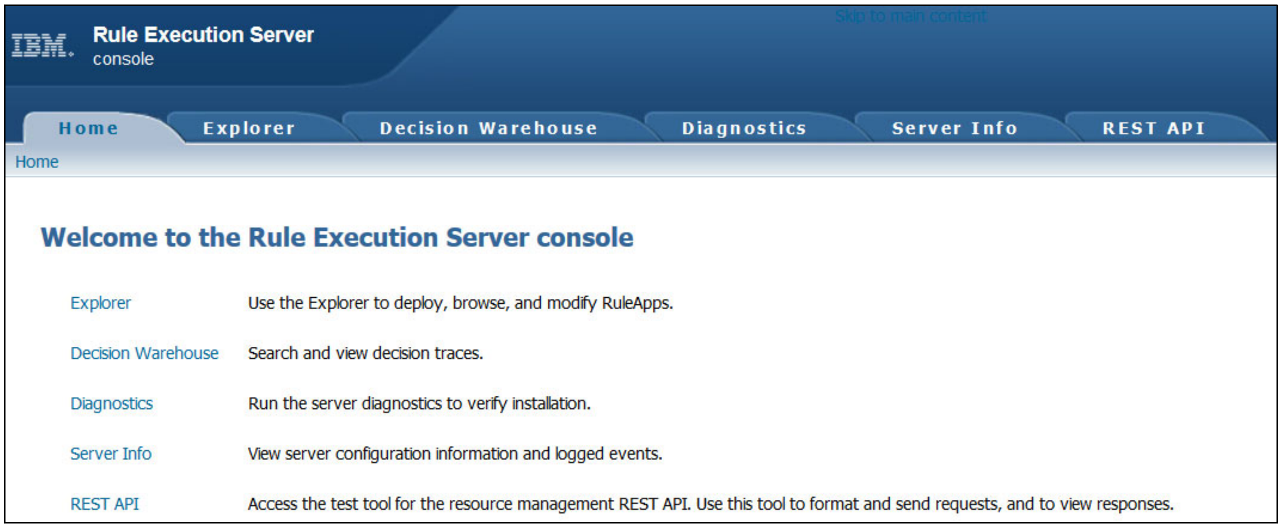
1. In a web browser, open the web console for zRule Execution Server for z/OS by using the following URL:

   `http://<your.server.address>:<PORT>/res`

2. At the login prompt for the Rule Execution Server console, enter the following login details:

   – Login: `resAdmin`
   – Password: `resAdmin`

3. On the Rule Execution Server console, click **Explorer** (Figure 3-64).

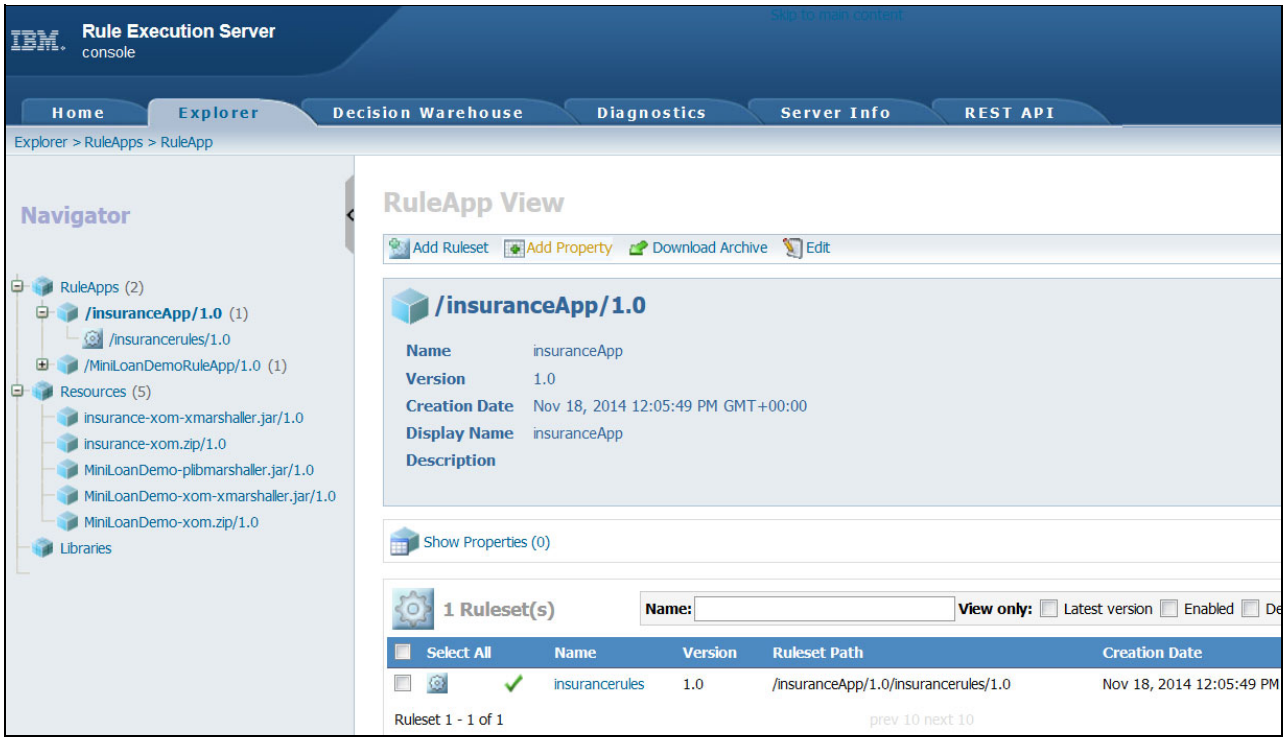


Welcome to the Rule Execution Server console

| | |
|---|---|
| Explorer | Use the Explorer to deploy, browse, and modify RuleApps. |
| Decision Warehouse | Search and view decision traces. |
| Diagnostics | Run the server diagnostics to verify installation. |
| Server Info | View server configuration information and logged events. |
| REST API | Access the test tool for the resource management REST API. Use this tool to format and send requests, and to view responses. |

*Figure 3-64 Exploring the rule project*

4. In the Navigator pane, click **RuleApps** to view the deployed RuleApp (Figure 3-65).



**/insuranceApp/1.0**

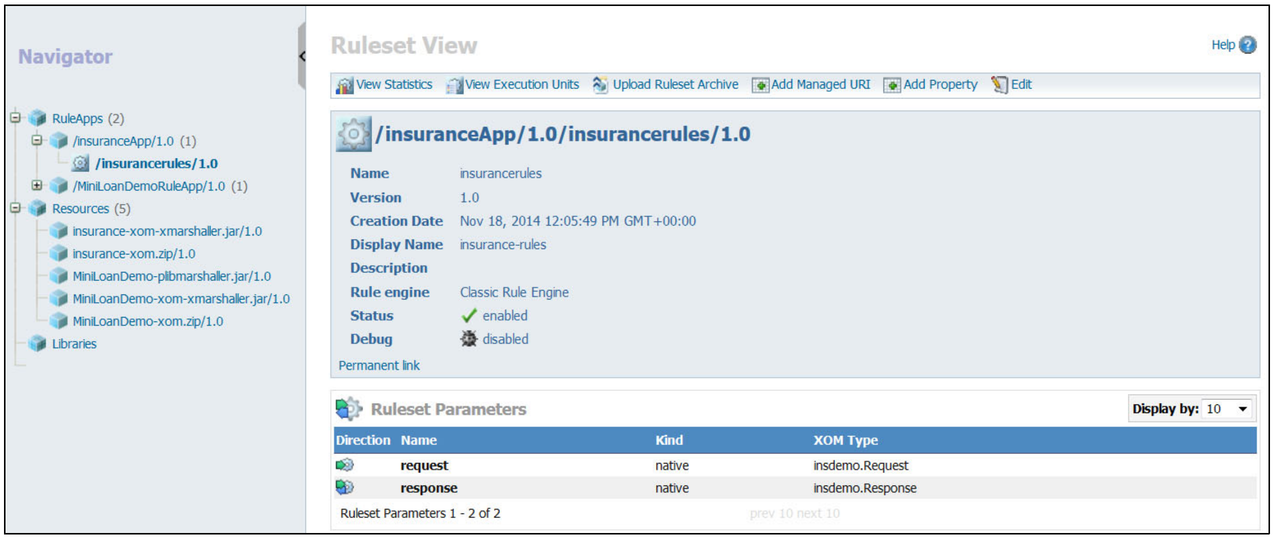
| | |
|---|---|
| Name | insuranceApp |
| Version | 1.0 |
| Creation Date | Nov 18, 2014 12:05:49 PM GMT+00:00 |
| Display Name | insuranceApp |
| Description | |

| Select All | Name | Version | Ruleset Path | Creation Date |
|---|---|---|---|---|
| ✓ | insurancerules | 1.0 | /insuranceApp/1.0/insurancerules/1.0 | Nov 18, 2014 12:05:49 PM |

Ruleset 1 - 1 of 1

*Figure 3-65 Viewing the deployed ruleset*

5. With the RulesApps tree fully expanded, click **/insurancerules/1.0** ruleset to see the Ruleset View (Figure 3-66).



**/insuranceApp/1.0/insurancerules/1.0**

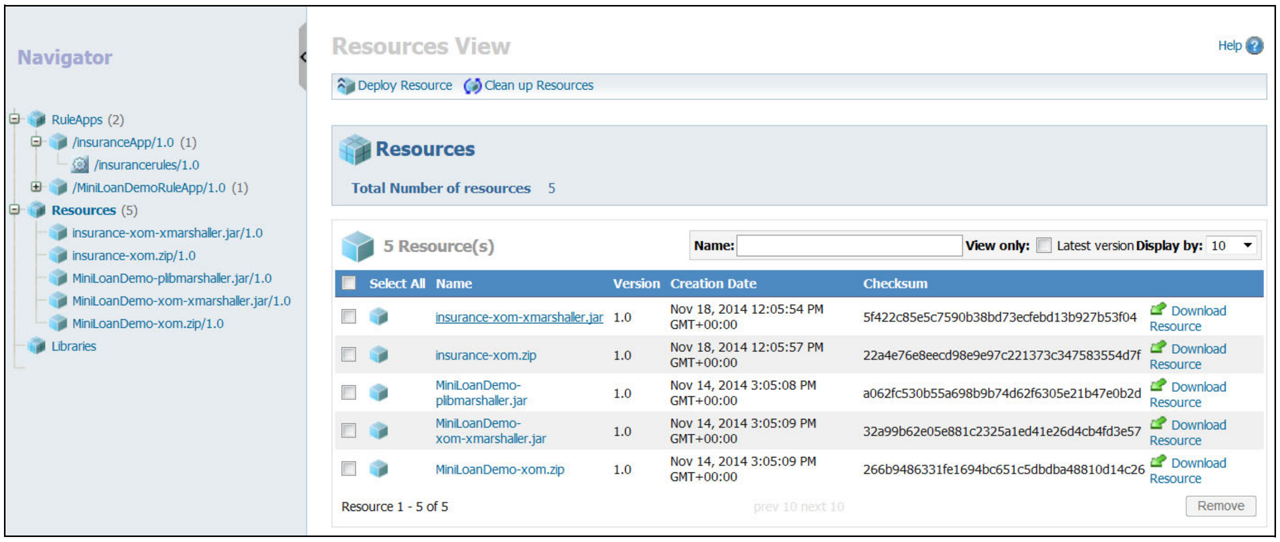
| | |
|---|---|
| Name | insurancerules |
| Version | 1.0 |
| Creation Date | Nov 18, 2014 12:05:49 PM GMT+00:00 |
| Display Name | insurance-rules |
| Description | |
| Rule engine | Classic Rule Engine |
| Status | ✓ enabled |
| Debug | disabled |

| Direction | Name | Kind | XOM Type |
|---|---|---|---|
| | request | native | insdemo.Request |
| | response | native | insdemo.Response |

Ruleset Parameters 1 - 2 of 2

*Figure 3-66 Viewing the deployed ruleset view*

6. In the Navigator pane, click **Resources** to view the deployed XOM and the marshaller file (Figure 3-67).



| Select All | Name | Version | Creation Date | Checksum | |
|---|---|---|---|---|---|
| | insurance-xom-xmarshaller.jar | 1.0 | Nov 18, 2014 12:05:54 PM GMT+00:00 | 5f422c85e5c7590b38bd73ecfebd13b927b53f04 | Download Resource |
| | insurance-xom.zip | 1.0 | Nov 18, 2014 12:05:57 PM GMT+00:00 | 22a4e76e8eecd98e9e97c221373c347583554d7f | Download Resource |
| | MiniLoanDemo-plibmarshaller.jar | 1.0 | Nov 14, 2014 3:05:08 PM GMT+00:00 | a062fc530b55a698b9b74d62f6305e21b47e0b2d | Download Resource |
| | MiniLoanDemo-xom-xmarshaller.jar | 1.0 | Nov 14, 2014 3:05:09 PM GMT+00:00 | 32a99b62e05e881c2325a1ed41e26d4cb4fd3e57 | Download Resource |
| | MiniLoanDemo-xom.zip | 1.0 | Nov 14, 2014 3:05:09 PM GMT+00:00 | 266b9486331fe1694bc651c5dbdba48810d14c26 | Download Resource |

Resource 1 - 5 of 5 prev 10 next 10

*Figure 3-67 Viewing deployed Java XOM and marshaller XOM*

## 3.2.10 Building a COBOL application for rule execution

To execute the rules, you call the ruleset from the COBOL application. You can use the zRule COBOL stub API to invoke the rule execution in a running instance of zRule Execution Server for z/OS.

To build the COBOL application, follow these steps:

1. Include the required copybooks for the zRule COBOL stub API:

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

2. Specify the ruleset path to initialize the values that are passed to zRule Execution Server for z/OS:

```
* ruleset path from the zRules Execution Server
  MOVE "/insuranceApp/insurancerules"  TO HBRA-CONN-RULEAPP-PATH
```

3. Configure the ruleset parameter:

   – Set the name of the parameter:

   ```
   MOVE 'request' TO HBRA-RA-PARAMETER-NAME(1)
   ```

   – Set the length of the parameter:

   ```
   MOVE LENGTH OF REQUEST TO HBRA-RA-DATA-LENGTH(1)
   ```

   – Set the address of the parameter:

   ```
   SET HBRA-RA-DATA-ADDRESS(1) TO ADDRESS OF REQUEST
   ```

4. Connect to zRule Execution Server for z/OS:

```
CALL 'HBRCONN' USING HBRA-CONN-AREA.
```

5. Execute the ruleset:

```
CALL 'HBRCONN' USING HBRA-CONN-AREA.
```

6. Disconnect from zRule Execution Server for z/OS:

```
CALL 'HBRDISC' USING HBRA-CONN-AREA.
```

### COBOL application sample

Example 3-4 includes a sample COBOL application that you can use to call the rules that you designed in the insurance-rules project.

**Additional resources:** You can find the `INSMAIN.cbl` application sample in the additional information that is included in this book in the `code/Chapter3/CopybookBased` directory. See Appendix C, "Additional material" on page 239.

*Example 3-4 COBOL application sample to call the rules on zRule Execution Server for z/OS*

```
IDENTIFICATION DIVISION.
       PROGRAM-ID. "INSMAIN".
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
           COPY INSDEMO.
       01 WS-REASON-CODES.
           COPY HBRC.
           COPY HBRWS.
       01 WS-MESSAGE-IDX      PIC 9(2).
       01 WS-MAX-TABLE-LEN    PIC 9(18).

       PROCEDURE DIVISION.
      * Init ruleset parameter data
           MOVE 'John'             TO FIRST-NAME
           MOVE 'Smith'            TO LAST-NAME
           MOVE 'XA123456'         TO ZIPCODE
           MOVE 123456             TO HOUSE-NUM
           MOVE 17                 TO AGE
           MOVE '20110908'         TO LIC-DATE
           MOVE 'F'                TO LIC-STATUS
           MOVE 4                  TO NUMBER-ACCIDENTS
           MOVE 'F'                TO APPROVED
           MOVE 100                TO BASE-PRICE
           MOVE 0                  TO MSG-COUNT
      * Move ruleset parameters to table HBRA-RA-PARMETERS
           MOVE ZERO               TO HBRA-CONN-RETURN-CODES
           MOVE LOW-VALUES         TO HBRA-RA-PARMETERS
           MOVE "/insuranceApp/insurancerules"
                                   TO HBRA-CONN-RULEAPP-PATH
      * Parameter Borrower
           MOVE LOW-VALUES         TO HBRA-RA-PARMETERS.
           MOVE 'request'          TO HBRA-RA-PARAMETER-NAME(1)
           MOVE LENGTH OF REQUEST  TO HBRA-RA-DATA-LENGTH(1)
           SET HBRA-RA-DATA-ADDRESS(1)
                                   TO ADDRESS OF REQUEST
      * Parameter Loan
           MOVE 'response'         TO HBRA-RA-PARAMETER-NAME(2)
```

```
        MOVE LENGTH OF RESPONSE TO HBRA-RA-DATA-LENGTH(2)
   * For ODO Table, the length represents the max length.
        COMPUTE WS-MAX-TABLE-LEN = LENGTH OF Messages * 100
        ADD WS-MAX-TABLE-LEN    TO HBRA-RA-DATA-LENGTH(2)
        SET HBRA-RA-DATA-ADDRESS(2)
                                TO ADDRESS OF RESPONSE
   * 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
   * Invoke rule execution server
        CALL 'HBRRULE' USING HBRA-CONN-AREA
        IF HBRA-CONN-COMPLETION-CODE NOT EQUAL HBR-CC-OK
            DISPLAY "invoke zRules failed"
            DISPLAY "CC code " HBRA-CONN-COMPLETION-CODE
            DISPLAY "RC code " HBRA-CONN-REASON-CODE
            DISPLAY "Message " HBRA-RESPONSE-MESSAGE
        ELSE
            DISPLAY 'invoke zRules successful'
        END-IF
   * Get disconnect to rule execution server
        CALL 'HBRDISC' USING HBRA-CONN-AREA
   * Display result
        DISPLAY         "********** EXECUTION RESULT *********"
        DISPLAY         "DRIVER NAME: " FIRST-NAME
        DISPLAY         "RESPONSE APPROVED: " APPROVED
        IF approved = "F"
           DISPLAY      "Reject messages:"
           PERFORM VARYING WS-MESSAGE-IDX FROM 1 BY 1
                           UNTIL WS-MESSAGE-IDX > MSG-COUNT
              DISPLAY messages (WS-MESSAGE-IDX)
           END-PERFORM
        END-IF
        DISPLAY         "*************************************"
  STOP RUN.
```

### Rule execution

You can compile and run the COBOL application on z/OS. In the COBOL application sample that is shown in Example 3-4 on page 65, you hardcoded the following input values:

- AGE: 17
- NUMBER-ACCIDENTS: 4

Example 3-5 shows the results after the rule execution.

*Example 3-5 Rule execution result*

```
********** EXECUTION RESULT *********
DRIVER NAME: John
RESPONSE APPROVED: F
Reject messages:
Accidents number exceeds the maximum
The age exceeds the maximum or minimum
**************************************
```

## 3.3 Getting started from an existing rule project

This section provides guidance about how to share business rules from an existing Java based rule project to a COBOL or PL/I application on z/OS. The following instructions describe the process for COBOL but equally apply when using PL/I.

### 3.3.1 Scenario overview

In this scenario, an insurance company has an existing business rule application to perform user validation for an insurance application. The rule projects, which the company currently uses, contain a BOM that is based on a Java XOM. The company deploys the rules to Rule Execution Server in a distributed environment.

The company now wants to share the Java rule projects with COBOL applications that run on z/OS and to manage the changes that are made to these rules. To share rules with COBOL applications, the company must add the necessary COBOL structures to the BOM and then generate a COBOL copybook. With these structures in the rule project, the company can then deploy the rules application to zRule Execution Server for z/OS so that the COBOL application can call the rulesets and execute the rules.

This section provides an existing rule project, `sharinginsurance-rules`, which has a BOM that is generated from a Java XOM (`sharinginsurance-xom`). It also uses a RuleApp project (`sharinginsuranceApp`) that is used for rule deployment to the runtime environment.

You can import the existing rule project from the source code that is delivered with this book. See Appendix C, “Additional material” on page 239 for details.

To import the example rule project, follow these steps:

1. From the Rule Explorer view, right-click and then select **Import** from the menu.
2. In the Import dialog, select **Existing Projects into Workspace**, as shown in Figure 3-68. Click **Next**.

**Tip:** You can quickly reduce the list of import sources by typing part of the name of the import source in the Select an import source field.

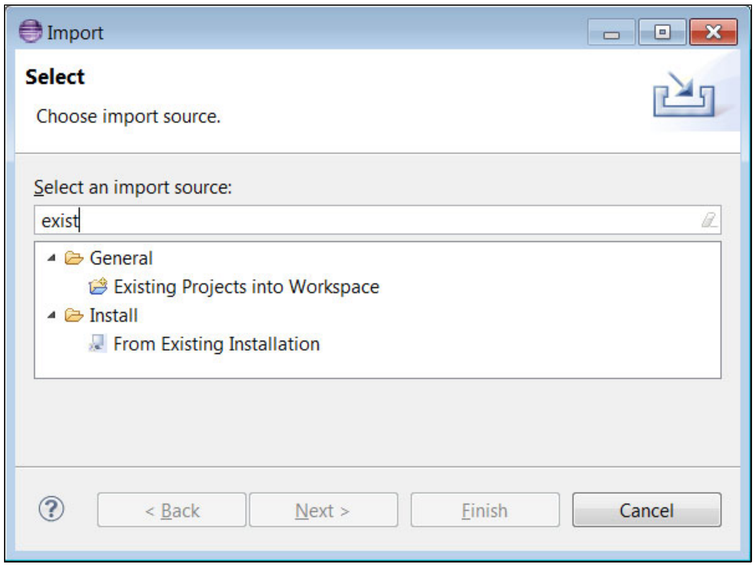
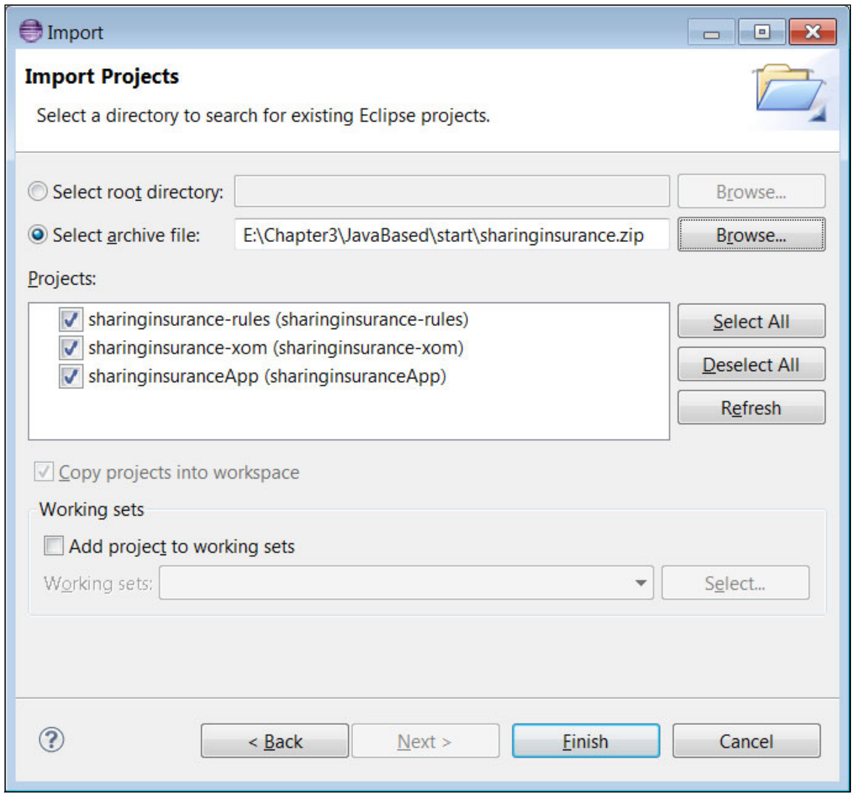


Figure 3-68 Importing existing projects

3. In the Import Projects dialog, select the **Select archive file** option and browse to the `sharinginsurance.zip` file. Select all three projects, as shown in Figure 3-69, and click **Finish**.


Import

**Import Projects**

Select a directory to search for existing Eclipse projects.

Select root directory: [ ] Browse...

(●) Select archive file: E:\Chapter3\JavaBased\start\sharinginsurance.zip Browse...

Projects:

- [✓] sharinginsurance-rules (sharinginsurance-rules)
- [✓] sharinginsurance-xom (sharinginsurance-xom)
- [✓] sharinginsuranceApp (sharinginsuranceApp)

[✓] Copy projects into workspace

Working sets

[ ] Add project to working sets

Working sets:

*Figure 3-69 Importing the insurance projects*

4. Figure 3-70 shows the existing rule project structure in Rule Designer.

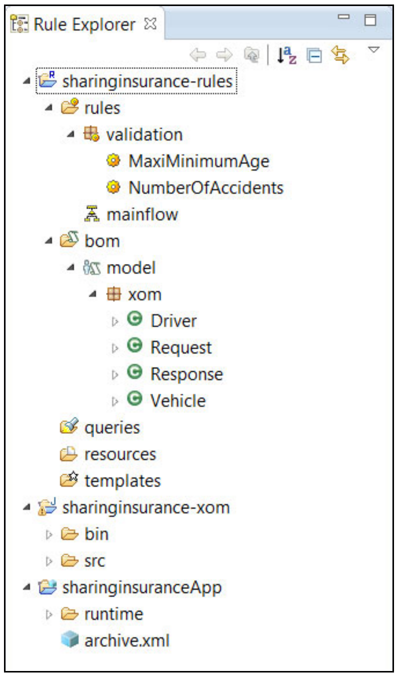


*Figure 3-70 Existing rule project structure*

## 3.3.2 Generating a copybook from the BOM

You use the COBOL Enabled BOM feature of Rule Designer to generate a copybook from the BOM in the existing rule project.

Use the following configuration for the BOM:

- Specify each Java class type that you want to use as top-level data items in the copybook.
- Enter a name for the runtime marshaller project and package, which are created during the copybook generation.

To configure the BOM for copybook generation, follow these steps:

1. In the Rule Explorer, right-click the **sharinginsurance-rules** rule project, and select **Properties** → **COBOL Management** → **COBOL Enabled BOM** (Figure 3-71). Click **Add**.

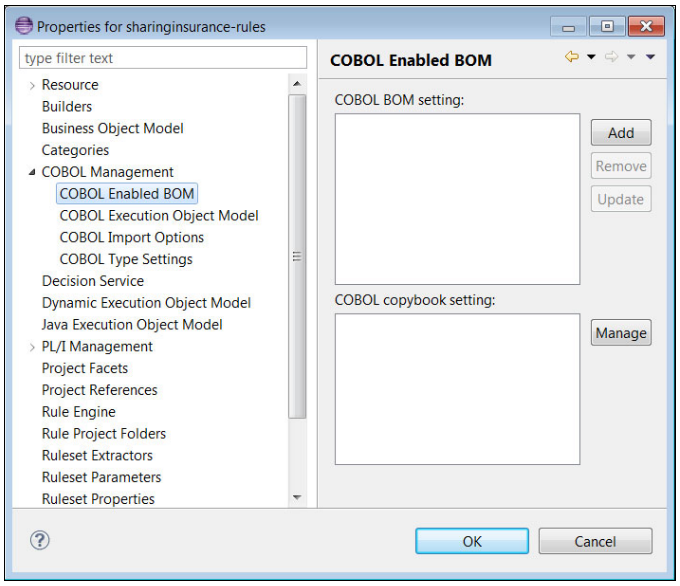


*Figure 3-71 Navigating to the COBOL enabled BOM*

2. In the Select BOM entry dialog, select **model**, and click **OK** (Figure 3-72).

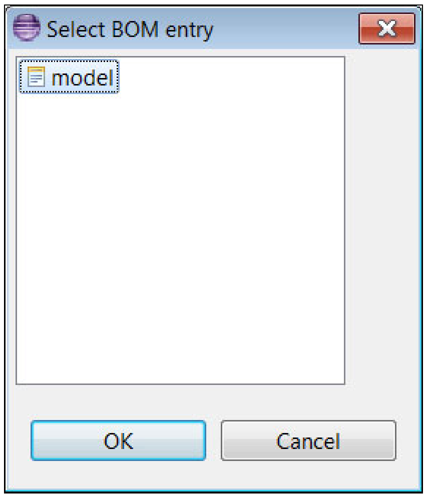
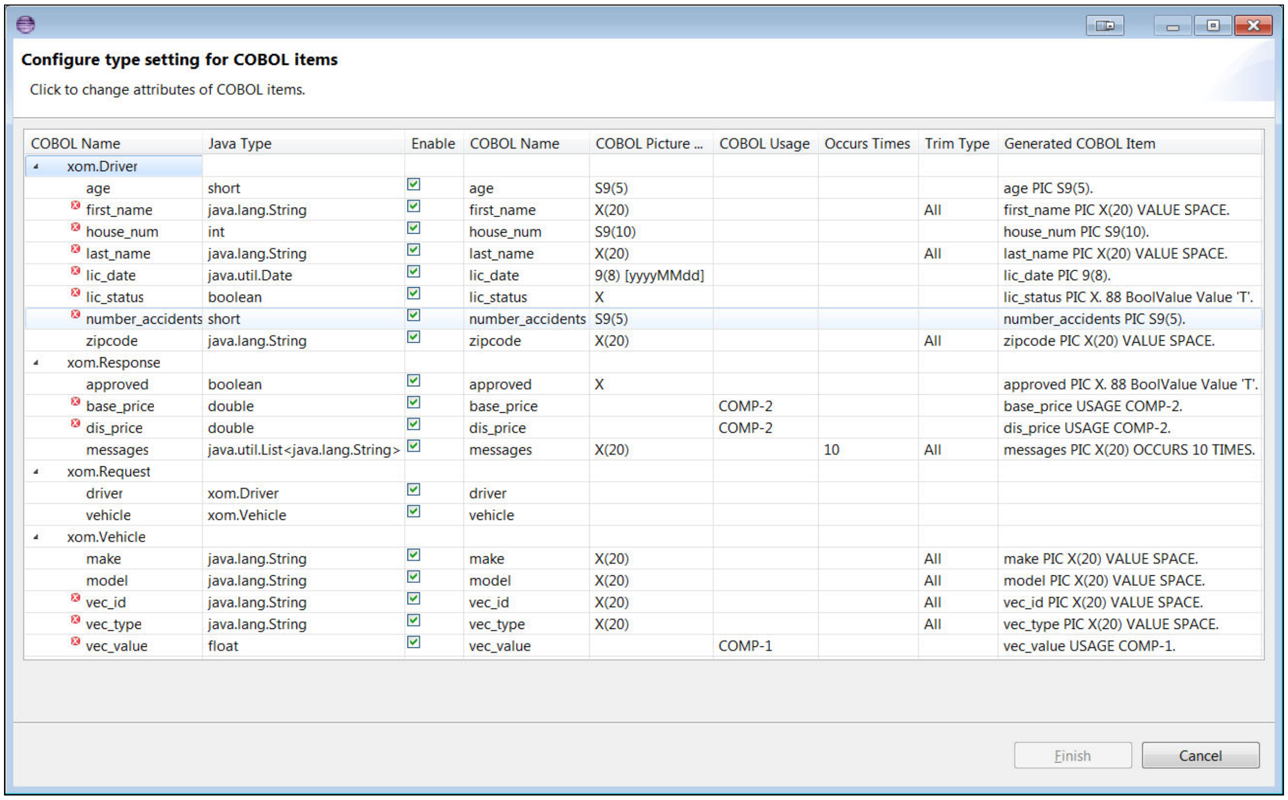


*Figure 3-72 Selecting the BOM model*

3. In the Resource Configuration section, accept the default names for the runtime Marshaller Project and Marshaller Package. Click **Next**.

A table shows the proposed mapping between the Java structures in the BOM and the COBOL structures, as shown in Figure 3-73.

**Important:** Several red error boxes show on certain lines. They are present because the Java attribute name is not a valid COBOL name. Those fields have not been enabled.



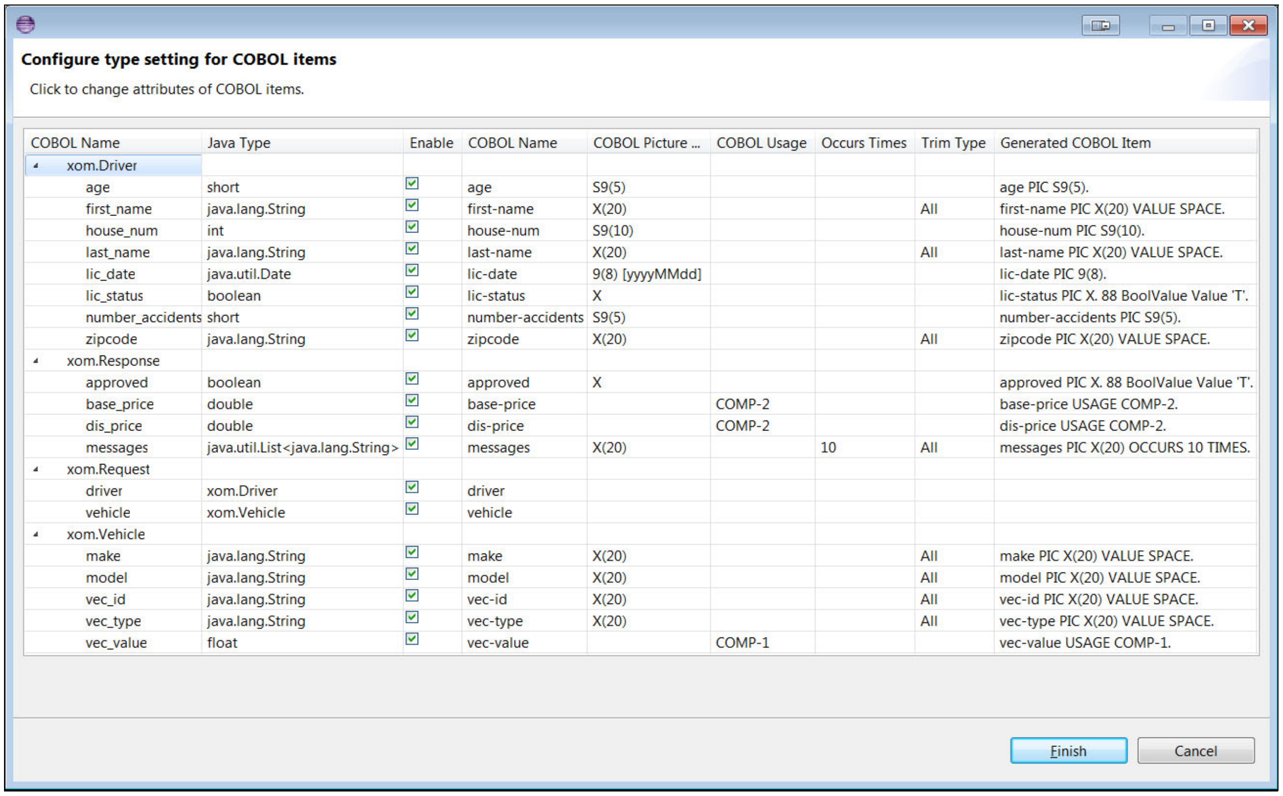
**Configure type setting for COBOL items**

Click to change attributes of COBOL items.

| COBOL Name | Java Type | Enable | COBOL Name | COBOL Picture ... | COBOL Usage | Occurs Times | Trim Type | Generated COBOL Item |
|---|---|---|---|---|---|---|---|---|
| xom.Driver | | | | | | | | |
| age | short | ✓ | age | S9(5) | | | | age PIC S9(5). |
| ⊗ first_name | java.lang.String | ✓ | first_name | X(20) | | | All | first_name PIC X(20) VALUE SPACE. |
| ⊗ house_num | int | ✓ | house_num | S9(10) | | | | house_num PIC S9(10). |
| ⊗ last_name | java.lang.String | ✓ | last_name | X(20) | | | All | last_name PIC X(20) VALUE SPACE. |
| ⊗ lic_date | java.util.Date | ✓ | lic_date | 9(8) [yyyyMMdd] | | | | lic_date PIC 9(8). |
| ⊗ lic_status | boolean | ✓ | lic_status | X | | | | lic_status PIC X. 88 BoolValue Value 'T'. |
| ⊗ number_accidents | short | ✓ | number_accidents | S9(5) | | | | number_accidents PIC S9(5). |
| zipcode | java.lang.String | ✓ | zipcode | X(20) | | | All | zipcode PIC X(20) VALUE SPACE. |
| xom.Response | | | | | | | | |
| approved | boolean | ✓ | approved | X | | | | approved PIC X. 88 BoolValue Value 'T'. |
| ⊗ base_price | double | ✓ | base_price | | COMP-2 | | | base_price USAGE COMP-2. |
| ⊗ dis_price | double | ✓ | dis_price | | COMP-2 | | | dis_price USAGE COMP-2. |
| messages | java.util.List<java.lang.String> | ✓ | messages | X(20) | | 10 | All | messages PIC X(20) OCCURS 10 TIMES. |
| xom.Request | | | | | | | | |
| driver | xom.Driver | ✓ | driver | | | | | |
| vehicle | xom.Vehicle | ✓ | vehicle | | | | | |
| xom.Vehicle | | | | | | | | |
| make | java.lang.String | ✓ | make | X(20) | | | All | make PIC X(20) VALUE SPACE. |
| model | java.lang.String | ✓ | model | X(20) | | | All | model PIC X(20) VALUE SPACE. |
| ⊗ vec_id | java.lang.String | ✓ | vec_id | X(20) | | | All | vec_id PIC X(20) VALUE SPACE. |
| ⊗ vec_type | java.lang.String | ✓ | vec_type | X(20) | | | All | vec_type PIC X(20) VALUE SPACE. |
| ⊗ vec_value | float | ✓ | vec_value | | COMP-1 | | | vec_value USAGE COMP-1. |

*Figure 3-73 Configuring the BOM to COBOL type mapping*

4. For each of the fields with an error, amend the COBOL name to make it suitable, for example, by converting each underscore (_) character to a dash (-) character (Figure 3-74).



**Configure type setting for COBOL items**

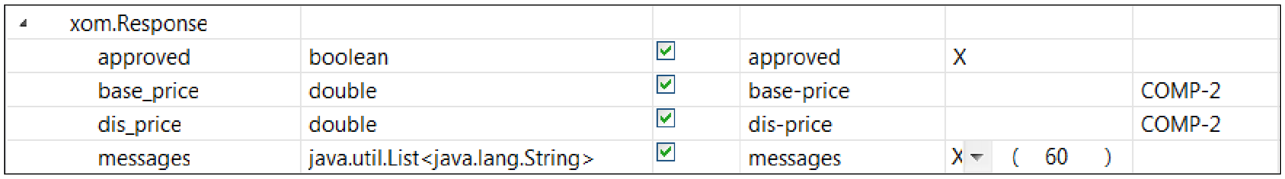
Click to change attributes of COBOL items.

| COBOL Name | Java Type | Enable | COBOL Name | COBOL Picture ... | COBOL Usage | Occurs Times | Trim Type | Generated COBOL Item |
|---|---|---|---|---|---|---|---|---|
| xom.Driver | | | | | | | | |
| age | short | ✓ | age | S9(5) | | | | age PIC S9(5). |
| first_name | java.lang.String | ✓ | first-name | X(20) | | | All | first-name PIC X(20) VALUE SPACE. |
| house_num | int | ✓ | house-num | S9(10) | | | | house-num PIC S9(10). |
| last_name | java.lang.String | ✓ | last-name | X(20) | | | All | last-name PIC X(20) VALUE SPACE. |
| lic_date | java.util.Date | ✓ | lic-date | 9(8) [yyyyMMdd] | | | | lic-date PIC 9(8). |
| lic_status | boolean | ✓ | lic-status | X | | | | lic-status PIC X. 88 BoolValue Value 'T'. |
| number_accidents | short | ✓ | number-accidents | S9(5) | | | | number-accidents PIC S9(5). |
| zipcode | java.lang.String | ✓ | zipcode | X(20) | | | All | zipcode PIC X(20) VALUE SPACE. |
| xom.Response | | | | | | | | |
| approved | boolean | ✓ | approved | X | | | | approved PIC X. 88 BoolValue Value 'T'. |
| base_price | double | ✓ | base-price | | COMP-2 | | | base-price USAGE COMP-2. |
| dis_price | double | ✓ | dis-price | | COMP-2 | | | dis-price USAGE COMP-2. |
| messages | java.util.List<java.lang.String> | ✓ | messages | X(20) | | 10 | All | messages PIC X(20) OCCURS 10 TIMES. |
| xom.Request | | | | | | | | |
| driver | xom.Driver | ✓ | driver | | | | | |
| vehicle | xom.Vehicle | ✓ | vehicle | | | | | |
| xom.Vehicle | | | | | | | | |
| make | java.lang.String | ✓ | make | X(20) | | | All | make PIC X(20) VALUE SPACE. |
| model | java.lang.String | ✓ | model | X(20) | | | All | model PIC X(20) VALUE SPACE. |
| vec_id | java.lang.String | ✓ | vec-id | X(20) | | | All | vec-id PIC X(20) VALUE SPACE. |
| vec_type | java.lang.String | ✓ | vec-type | X(20) | | | All | vec-type PIC X(20) VALUE SPACE. |
| vec_value | float | ✓ | vec-value | | COMP-1 | | | vec-value USAGE COMP-1. |

Figure 3-74 Changing the COBOL names

5. Click the **COBOL Picture** field for the messages item of the xom.Response class and change the default length for messages from X(20) to X(60), as shown in Figure 3-75. Click **Finish**.

**Important:** The default mapping from Java String to COBOL Picture length is 20. You adjust this value per rule project. In this scenario, the sharinginsurance-rules project uses a COBOL Picture length mapping value of 60, as required by the real reject message in the business rules.



| | | | | | |
|---|---|---|---|---|---|
| xom.Response | | | | | |
| approved | boolean | ✓ | approved | X | |
| base_price | double | ✓ | base-price | | COMP-2 |
| dis_price | double | ✓ | dis-price | | COMP-2 |
| messages | java.util.List<java.lang.String> | ✓ | messages | X ( 60 ) | |

Figure 3-75 Changing the default mapping of the message item of the xom.Response class

6. The BOM model is now listed as a COBOL enabled BOM, as shown in Figure 3-76. Click **Manage**.

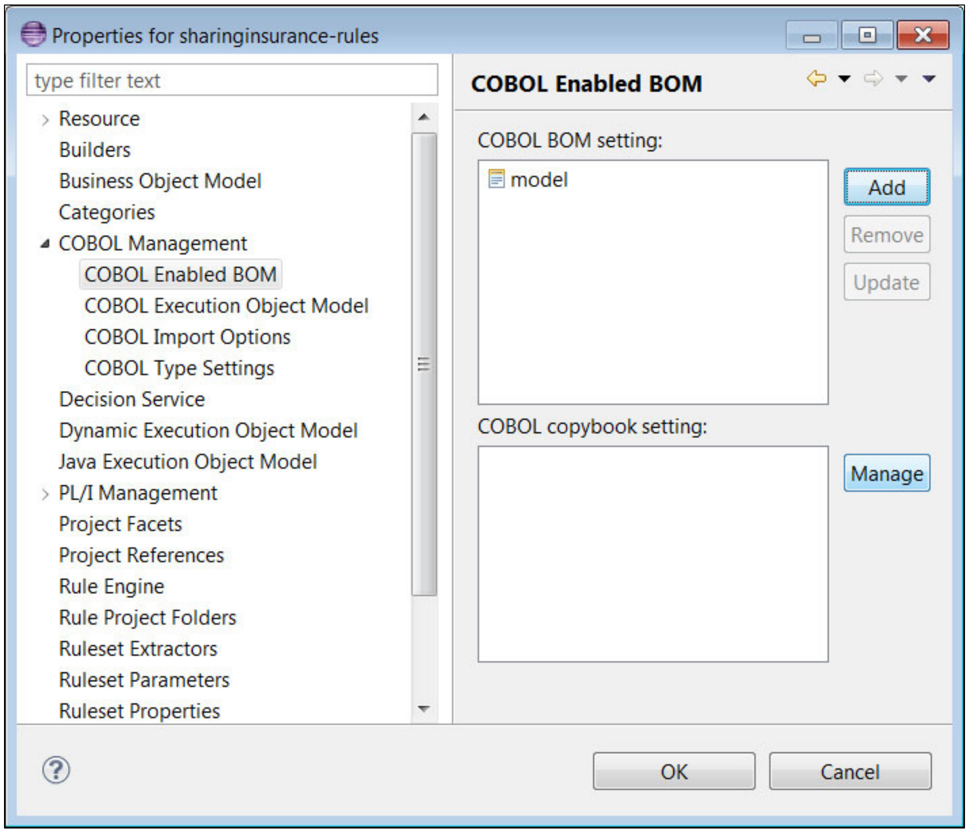
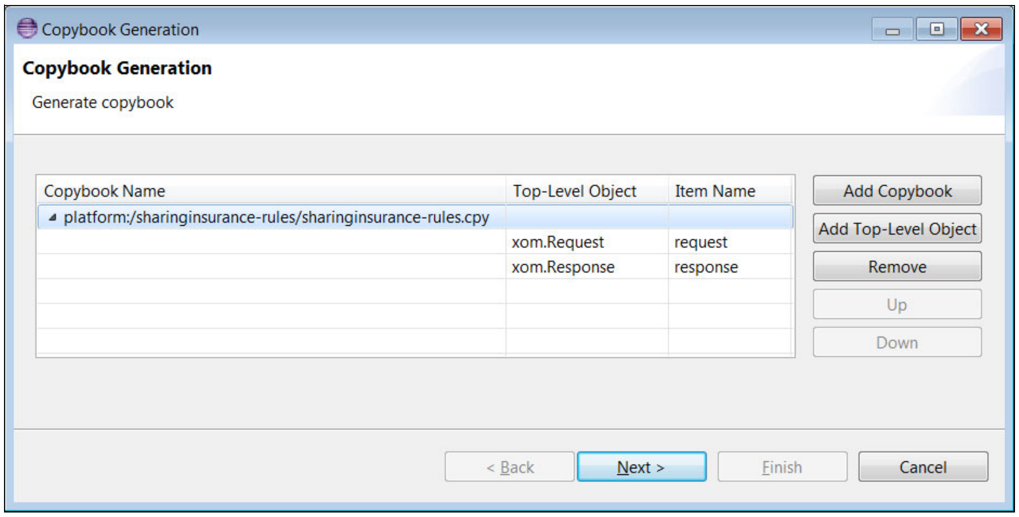


Figure 3-76 COBOL enabled BOM

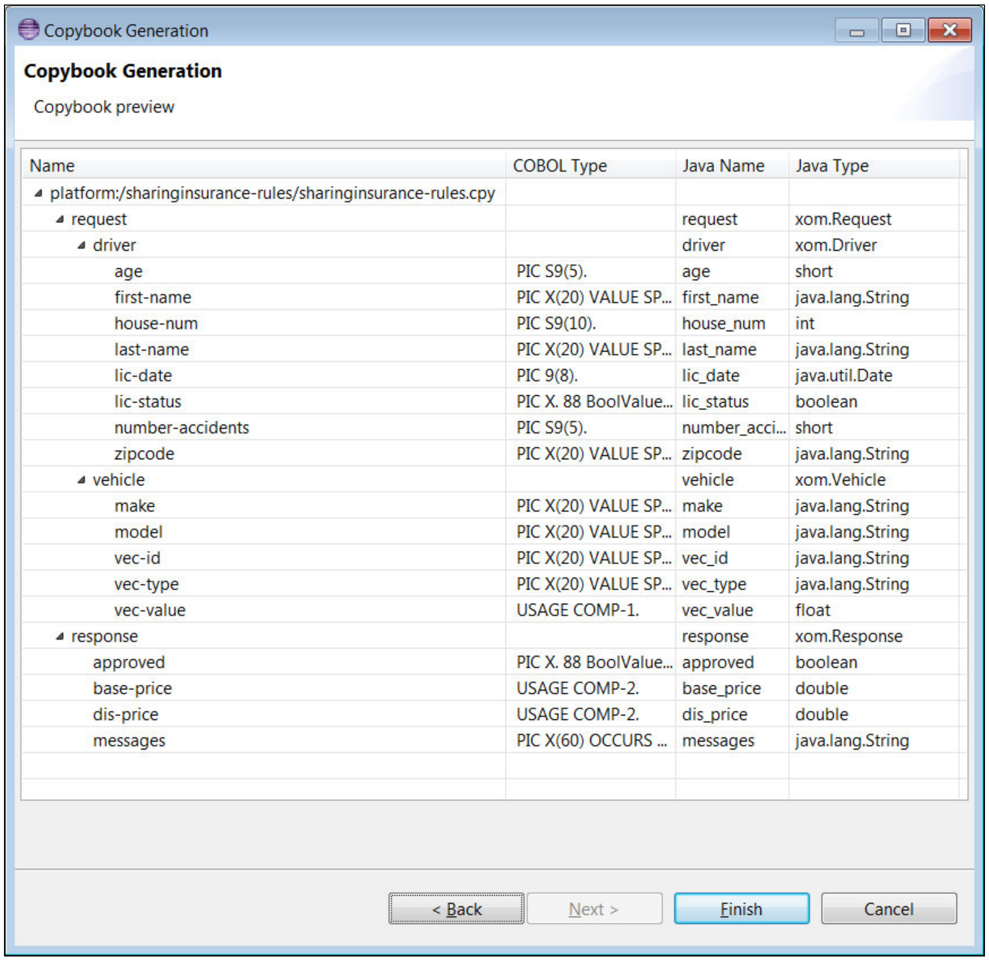
7. In the Copybook Generation dialog (Figure 3-77), review the information and click **Next**.



| Copybook Name | Top-Level Object | Item Name |
|---|---|---|
| platform:/sharinginsurance-rules/sharinginsurance-rules.cpy | | |
| | xom.Request | request |
| | xom.Response | response |

Figure 3-77 Copybook generation information

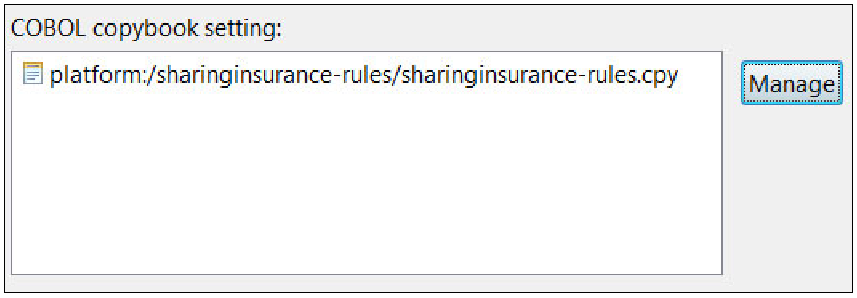
8. You see the Copybook Generation preview dialog, as shown in Figure 3-78. Review the information and click **Finish**.



| Name | COBOL Type | Java Name | Java Type |
|---|---|---|---|
| platform:/sharinginsurance-rules/sharinginsurance-rules.cpy | | | |
| request | | request | xom.Request |
| driver | | driver | xom.Driver |
| age | PIC S9(5). | age | short |
| first-name | PIC X(20) VALUE SP... | first_name | java.lang.String |
| house-num | PIC S9(10). | house_num | int |
| last-name | PIC X(20) VALUE SP... | last_name | java.lang.String |
| lic-date | PIC 9(8). | lic_date | java.util.Date |
| lic-status | PIC X. 88 BoolValue... | lic_status | boolean |
| number-accidents | PIC S9(5). | number_acci... | short |
| zipcode | PIC X(20) VALUE SP... | zipcode | java.lang.String |
| vehicle | | vehicle | xom.Vehicle |
| make | PIC X(20) VALUE SP... | make | java.lang.String |
| model | PIC X(20) VALUE SP... | model | java.lang.String |
| vec-id | PIC X(20) VALUE SP... | vec_id | java.lang.String |
| vec-type | PIC X(20) VALUE SP... | vec_type | java.lang.String |
| vec-value | USAGE COMP-1. | vec_value | float |
| response | | response | xom.Response |
| approved | PIC X. 88 BoolValue... | approved | boolean |
| base-price | USAGE COMP-2. | base_price | double |
| dis-price | USAGE COMP-2. | dis_price | double |
| messages | PIC X(60) OCCURS ... | messages | java.lang.String |

*Figure 3-78 Copybook generation preview*

9. Returning to the COBOL Enabled BOM dialog, you see that a new copybook has been created in the COBOL copybook setting section (Figure 3-79). Click **OK**.



platform:/sharinginsurance-rules/sharinginsurance-rules.cpy

*Figure 3-79 COBOL copybook settings*

You can view the generated copybook, as shown in Figure 3-80:

`sharinginsurance-rules.cpy` generated copybook

**Important:** Do not change the generated copybook. When a change occurs to the BOM of the rule project, use the COBOL enabled BOM feature to update the copybook.

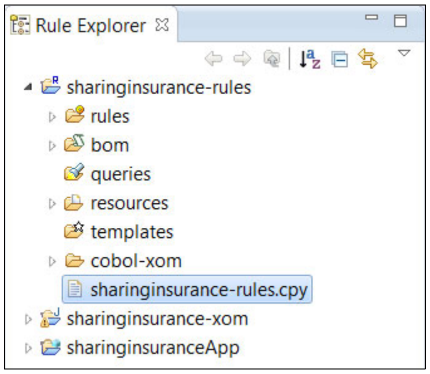


*Figure 3-80 Generated copybook*

## 3.3.3 Deploying rule artifacts to zRule Execution Server for z/OS

To execute a ruleset with zRule Execution Server for z/OS, you must deploy the rule project and the Java XOM to a zRule Execution Server for z/OS. The deployment process is the same as the process that is described in 3.2, “Getting started from a COBOL copybook” on page 25.

**Important:** Ensure that you start zRule Execution Server for z/OS successfully before you attempt to deploy rules.

To deploy rule artifacts to zRule Execution Server for z/OS, follow these steps:

1. Deploy the sharinginsuranceApp by using one of the following options:
   – Opening the project and double-clicking the **archive.xml** file to open the RuleApp editor. Then, in the Deployment pane of the RuleApp editor, click **Deploy**.
   – Using the menu options by right-clicking the project and selecting **RuleApp** → **Deploy**, as shown in Figure 3-81.

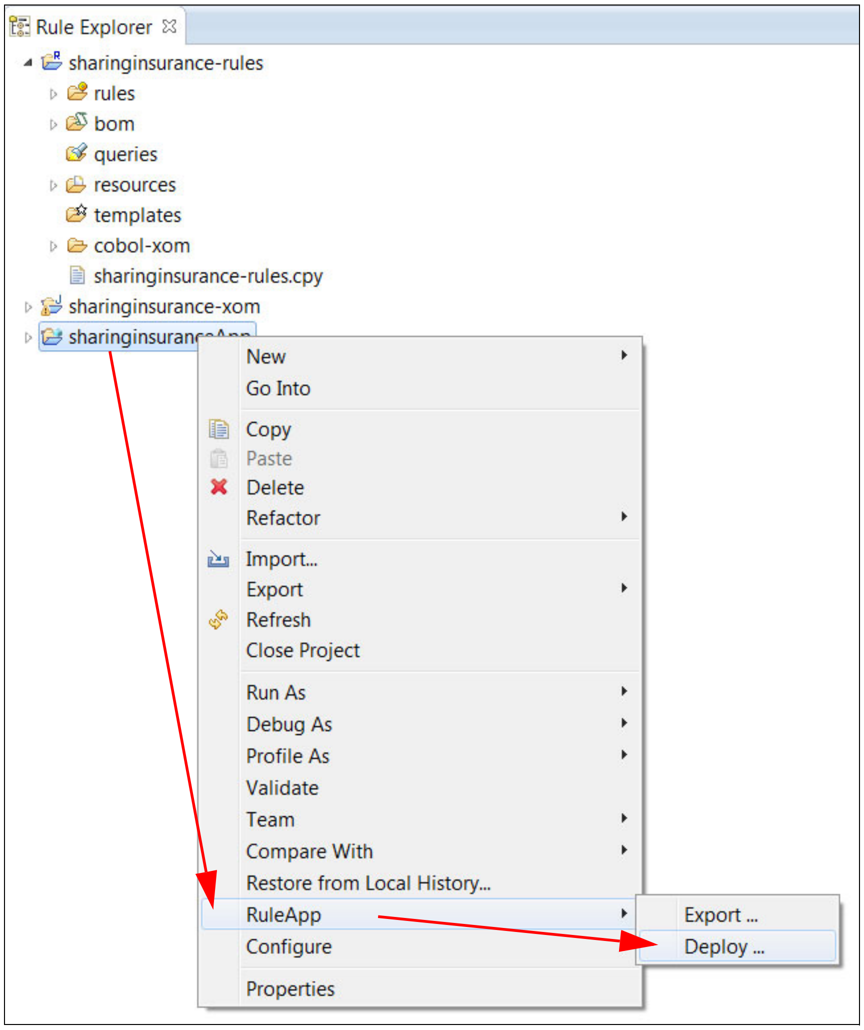


*Figure 3-81 Deploying the RuleApp*

2. For the deployment type, accept the **Increment RuleApp major version** default option, and click **Next**.
3. Select **Create a temporary Rule Execution Server configuration**, and enter the following details:
   – URL: `http://<your.server.address>:<PORT>/res`
   – Login: `resAdmin`
   – Password: `resAdmin`

   Click **Finish**.

Your artifacts are deployed to zRule Execution Server for z/OS. You can now build a COBOL application to invoke the rule execution.

## 3.3.4 Building a COBOL application for rule execution

You deployed the rule artifacts to zRule Execution Server for z/OS. You can now use the generated copybook to build a COBOL application for rule execution on z/OS.

**Additional resources:** You can find the examples that are used in this section in the additional information that is included in this book. For details, see Appendix C, “Additional material” on page 239.

### Generated copybook example

First, you must know the structure of the generated copybook, as shown in Example 3-6.

*Example 3-6 Generated copybook INSSHAR.cpy*

```
01 request.
        02 driver.
            03 age pic S9(5).
            03 first-name pic X(20) value SPACE.
            03 house-num pic S9(10).
            03 last-name pic X(20) value SPACE.
            03 lic-date pic 9(8).
            03 lic-status pic X.
                88 BoolValue value 'T'.
            03 number-accidents pic S9(5).
            03 zipcode pic X(20) value SPACE.
        02 vehicle.
            03 make pic X(20) value SPACE.
            03 model pic X(20) value SPACE.
            03 vec-id pic X(20) value SPACE.
            03 vec-type pic X(20) value SPACE.
            03 vec-value usage COMP-1.
    01 response.
        02 approved pic X.
            88 BoolValue value 'T'.
        02 base-price usage COMP-2.
        02 dis-price usage COMP-2.
        02 messages-Num pic 9(9).
        02 messages pic X(60) value SPACE Occurs 10 Times.
```

### COBOL application example

Now, you can build a COBOL application according to the generated copybook, as shown in Example 3-7.

*Example 3-7 COBOL application INSSHAR.cbl*

```
IDENTIFICATION DIVISION.
       PROGRAM-ID. "INSSHAR".
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      * include the generated copybook
           COPY INSSHAR.
       01 WS-REASON-CODES.
           COPY HBRC.
           COPY HBRWS.
      PROCEDURE DIVISION.
     * Init ruleset parameter data
          MOVE 'John'            TO FIRST-NAME
          MOVE 17                TO AGE
          MOVE 4                 TO NUMBER-ACCIDENTS
      ......
     * Move ruleset path to table HBRA-RA-PARMETERS
     ......
          MOVE "/sharinginsuranceApp/sharinginsurancerules"
                                 TO HBRA-CONN-RULEAPP-PATH
     * move ruleset parameter for request and response
          MOVE 'request'         TO HBRA-RA-PARAMETER-NAME(1)
     ......
          MOVE 'response'        TO HBRA-RA-PARAMETER-NAME(2)
       ......
     * Get connection to rule execution server
          CALL 'HBRCONN' USING HBRA-CONN-AREA.
     ......
     * Invoke rule execution server
          CALL 'HBRRULE' USING HBRA-CONN-AREA
     ......
     * Get disconnect to rule execution server
          CALL 'HBRDISC' USING HBRA-CONN-AREA
     ......
     * Display result
          DISPLAY        "RESPONSE APPROVED: " APPROVED
     ......
    STOP RUN.
```

### Rule execution result

You can compile and run the COBOL application on z/OS. The COBOL application example in Example 3-7 on page 78 produces the results that are shown in Example 3-8.

*Example 3-8 Results of compiling and running the COBOL application*

```
********** EXECUTION RESULT *********
DRIVER NAME: John
RESPONSE APPROVED: F
Reject messages:
Accidents number exceeds the maximum
The age exceeds the maximum or minimum
**************************************
```

4

# Managing business decisions through the full lifecycle

This chapter looks at the lifecycle of a decision and discusses considerations when deploying to a z/OS environment.

The following topics are covered in this chapter:

## 4.1 What is the lifecycle of rule artifacts in decisions

Rule artifacts within Operational Decision Manager follow a general lifecycle that can be tailored to the particular setup of your system. Rules can pass through one or more of the available tools and through one or more repositories or databases before being deployed to the production server. Your lifecycle can contain development, testing, production, and maintenance systems, which are maintained by developers and business users as appropriate.

A typical lifecycle includes rulesets and flows that are being developed by the development department and saved in a central repository. Specific rules can then be accessed and maintained by business users without the necessity of returning the rules to development to be implemented in the software. From this point, rule artifacts are tested and then can be deployed to the main production system. Maintenance can be carried out either by the business users (for rules modifications) or by development (for more substantial changes). A diagram of this process is shown in Figure 4-1.

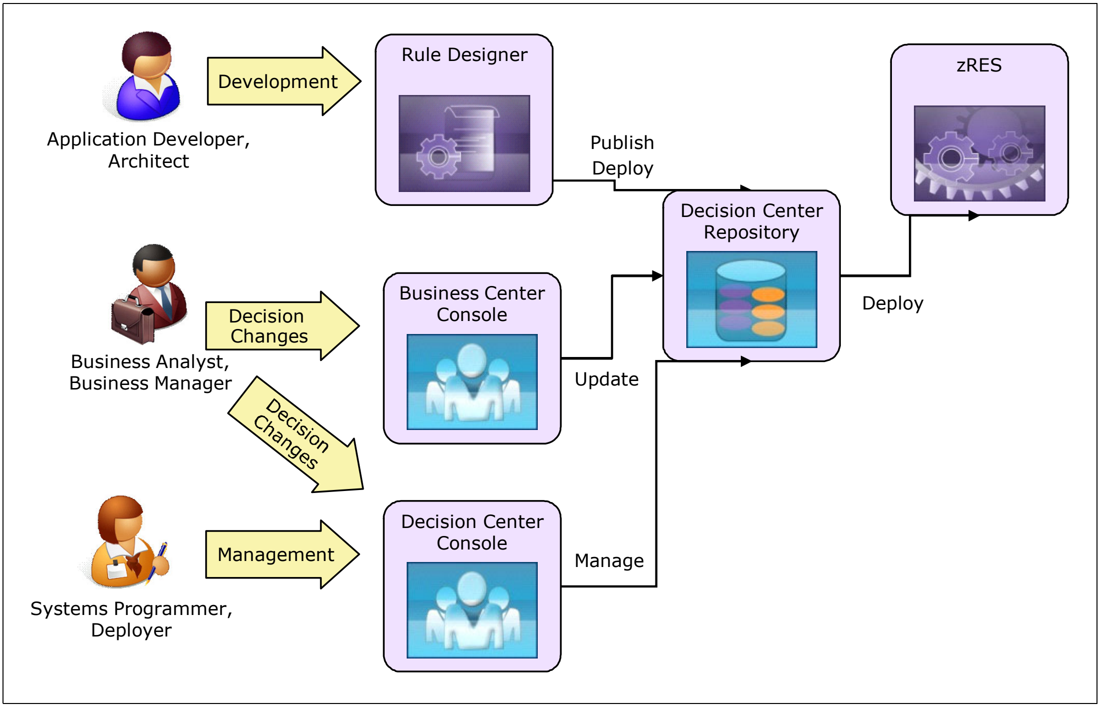


*Figure 4-1 Users and management of rules*

One of the key reasons for extracting decisions into a decision management system is that it allows you to separate the lifecycle of the decisions from the lifecycle of the application that invokes the decision. This separation is important particularly on z Systems where the application deployment cycles are often long, and the business wants to be able to change the business behavior of the application more quickly. By separating the decision lifecycle from the application lifecycle, you can gain agility in your business applications without sacrificing the reliability of the core logic of your business applications.

## 4.2 Working with rules through the lifecycle

The decision lifecycle starts with the initial location of the rules within the business applications. After the rules are identified, the data that is associated with the decisions can be extracted and a copybook can be created that contains the required information.

The business application must be refactored to remove the current implementation of the business decision and replace it with calls out to the Rule Execution Server for z/OS (zRES). The application might have decision logic and business logic interspersed throughout it. To optimize the application, refactor the application so that it makes the fewest calls possible to the zRES, as shown in Figure 4-2.

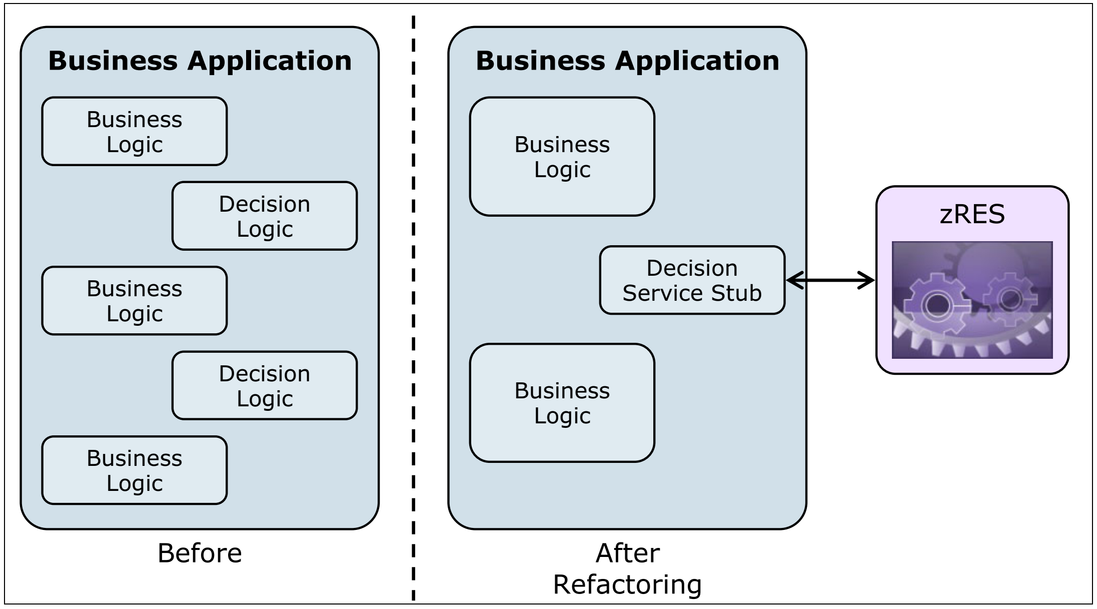


*Figure 4-2 Refactoring the business application*

After the copybook containing the required data for the decision is created or identified, the business rules for the initial version of the business decision can be authored. The initial creation of the business object model (BOM) and verbalization is done in Rule Designer. This action is generally considered an IT project, because it requires knowledge of the COBOL data structure.

After the BOM is created, the IT department typically creates the initial version of the business rules for the decision based on the rules that were previously embedded in the application. After the first version of the business rules is created, the decision can be deployed to a zRES environment and tested with the application.

With the first version of the decision now deployed, you must decide how to manage the ongoing lifecycle of the decision. There are multiple approaches to this problem. In certain development shops, the management of the business decisions remains a purely IT-based process, using the Rule Designer environment for managing and maintaining the decisions. Now that the business rules from the decision are authored in a far more accessible language, it is often advantageous to allow the business team to interact directly with the authored rules.

Business users can view and modify rules using the Decision Center Enterprise Console or Business Console as shown in Figure 4-3. Testing can still be carried out in the Decision Center without needing to involve the IT department.

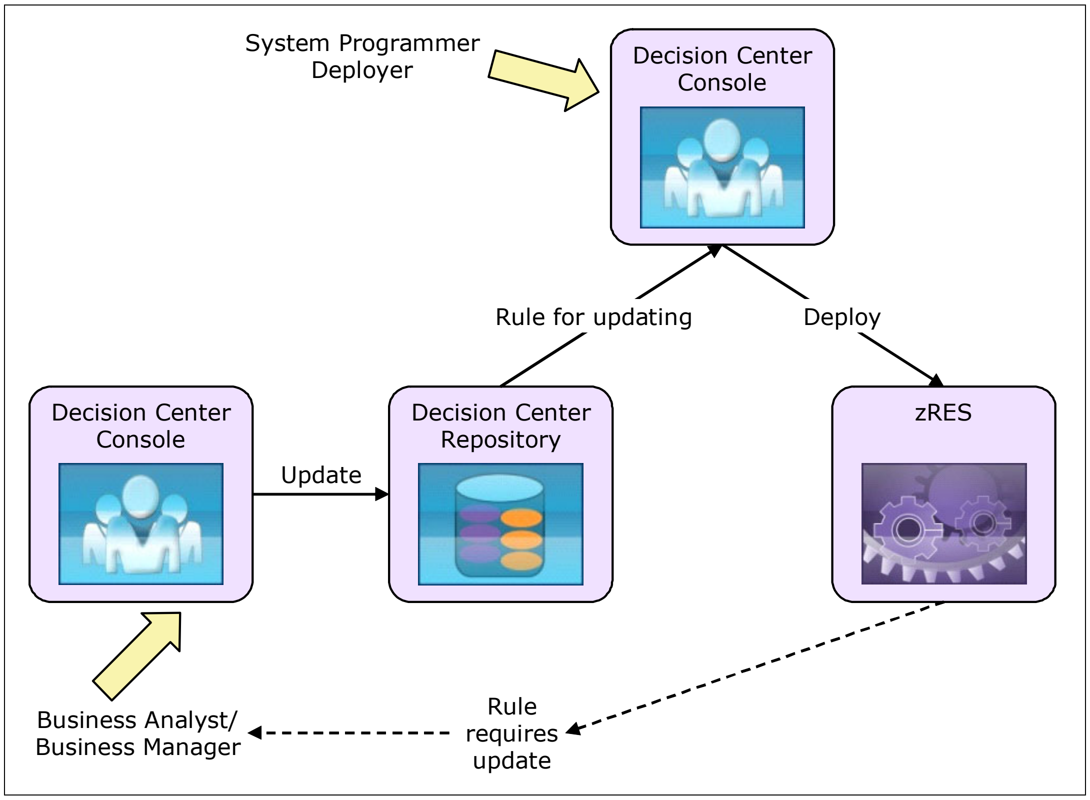


*Figure 4-3 Movement of a rule through the lifecycle*

After the initial deployment of the rules, the application developer can also publish the rules to the Decision Center repository. Other roles can become involved in the decision lifecycle.

The Decision Center consoles provide two distinct functions:

- The ability to change the rules of decisions that are published to the repository
- The ability to manage versions and deploy decisions from the repository to the rule runtime environments

An important part of the decision lifecycle is the ability to test the changes to the decision before deploying it to the final server. You can use Decision Center to define scenarios or test cases using, for example, a spreadsheet format to define the input and expected output. Decision Center can then take this data, deploy the ruleset to a configured zRES, and execute the decisions based on the supplied data. Decision Center can report situations where expected results are not returned. This function gives you the ability for the decision lifecycle to happen completely in isolation from the application lifecycle. Changes to the decision behavior can be tested in isolation from the application before they are deployed into the production system.

## 4.2.1 Managing artifacts

There are a number of artifacts in the decision lifecycle that require managing, as shown in Figure 4-4:

- Ruleset
- Java execution object module (XOM)
- RuleApp

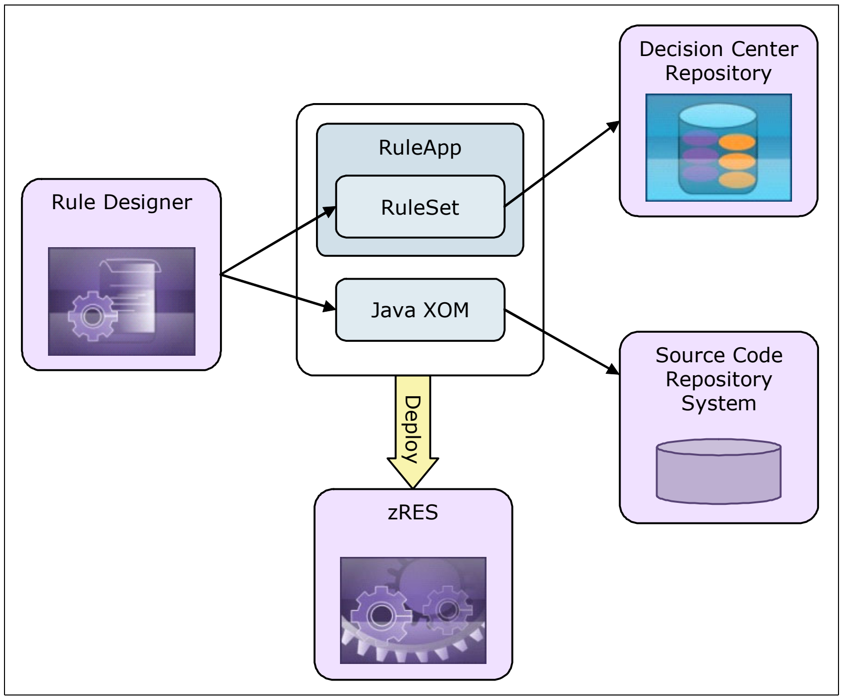


*Figure 4-4 Artifacts within the decision lifecycle*

### Ruleset

The ruleset contains a number of elements:

- The BOM
- The verbalization of that model
- The authored rules from the decision
- The ruleflow that guides the execution of the decision
- The declaration of the required parameters for this decision

The ruleset is the primary artifact that requires management. It contains the rules themselves and the BOM on which they are based. The *ruleset* is the artifact that is published to Decision Center. You can access the Decision Center to change the rules within a decision.

A ruleset can inherit from another ruleset. In this case, the decision contains the BOM, verbalization, and rules from both rulesets. This approach is a useful way to reuse rules that are shared across multiple decisions and manage the changes to the shared rules in a single project.

The name of a ruleset is significant. It forms part of the ruleset path that is used by the client to identify the decision that it wants to invoke on the server.

Any changes to the rules within a decision require the ruleset to be redeployed to make the new decision behavior available. A preferred practice is to increment the minor version of the decision when redeploying behavioral changes in the decision.

### Java XOM

The *Java execution object module (XOM)* is standard Java code that is the Java representation of the imported COBOL copybook or PL/I include file. zRES uses this Java XOM at run time for mapping the COBOL or PL/I data structure before rules can be executed. The Java XOM can be deployed to the zRES server directly from Rule Designer or by using JCL on the server's LPAR.

Because the code is standard Java code, it must be managed and maintained by using a source code management system. Rule Designer is based on Eclipse. Many of the standard source code management systems have plug-ins that allow you to manage and handle versioning for the Eclipse Java project directly from the Rule Designer environment. If the starting point for the project is an imported copybook, this code is generated by Rule Designer and must not be edited. If possible, mark this code as read-only within the source code management system.

Only deploy the Java XOM if there are changes to the underlying data structures that define the interface to this decision. This situation occurs when a change is made to the COBOL copybook that was imported to create the BOM. A change of this nature also requires a corresponding change in the COBOL client applications to use the new copybook structure. For this reason, changes to the COBOL copybook are considered an IT project and happen less frequently than changes to the decision behavior in the rules. A preferred practice is to increment the major version number of the decision when making these interface changes.

### RuleApp

The *RuleApp* is the deployment container for one or a number of related rulesets. RuleApps are created for deployment either within Rule Designer or Decision Center. They are a compressed (`.zip`) file that contains the required artifacts to execute the decision.

A RuleApp can be deployed directly to a server from either Rule Designer or Decision Center. Or, a RuleApp can be exported as a JAR file that can be managed externally to the Operational Decision Manager tool and deployed to a server by using scripts. This approach can be useful when defining the process of performing decision updates where there is no access from Rule Designer or Decision Center to the production zRES for z/OS.

The name of a RuleApp is significant. The name forms part of the ruleset path that is used by the client to identify the decision that it wants to invoke on the server.

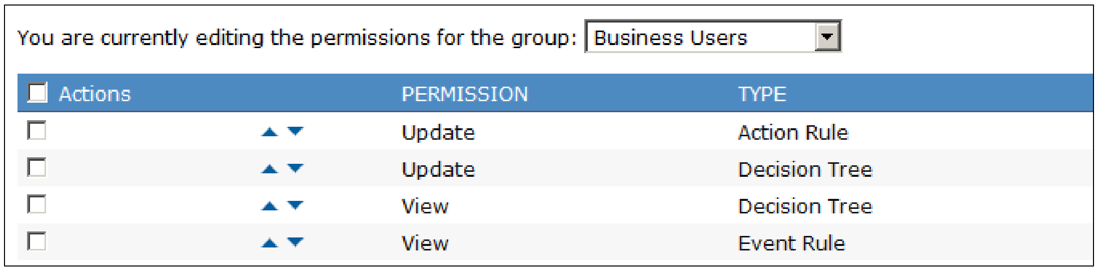
## 4.2.2 What roles are involved in the decision lifecycle

There are three roles that are involved in the lifecycle of a decision. The names vary from company to company, but there are normally people who can be attributed to one or more of the following roles:

- Application/decision developer
- Systems administrator/programmer
- Business team member

The major interaction among the team members occurs in the Decision Center environment. Here, a developer synchronizes the RuleApps on which the developer is working in Rule Designer. The systems administrator goes to Rule Designer to version and deploy decisions. The business team accesses Rule Designer to view or change decisions.

The Decision Center environment provides role-based access authorities to allow the systems administrator to give people the correct authority to access the rulesets for which they are responsible. The granularity of access authority is delivered at the ruleset level. But, with a little customizing, the granularity of management can be changed to match whatever is required by the organization. Figure 4-5 shows how a set of permissions might appear for a particular group of users.



You are currently editing the permissions for the group: Business Users

| ☐ Actions | | PERMISSION | TYPE |
|---|---|---|---|
| ☐ | ▲ ▼ | Update | Action Rule |
| ☐ | ▲ ▼ | Update | Decision Tree |
| ☐ | ▲ ▼ | View | Decision Tree |
| ☐ | ▲ ▼ | View | Event Rule |

*Figure 4-5 Example permission settings for a business user*

## Rule developer

The rule developer is generally an IT-based person. In most organizations, the rule developer is also part of the application development team that is responsible for the application that is being modernized by having its decisions extracted. In a larger organization, a dedicated team with the skill to externalize and develop decisions might exist, outside of the application development team.

The rule developer is responsible for creating the initial version of the BOM and the verbalization of that model. The rule developer normally writes the first pass at the rules based on what currently exists in the application code. The rule developer’s primary tool is Rule Designer. The rule developer is responsible for publishing the ruleset and the XOM to Decision Center. This person also ensures that the Java XOM code is maintained in a source code management system.

## Systems administrator

The systems administrator is responsible for the production zRES servers and their health. In the rule lifecycle, the systems administrator generally is responsible for versioning and deploying new versions of a decision into the zRES. The systems administrator is also responsible for maintaining the security model within the Decision Center environment to ensure that users have access to only the rulesets for which they are responsible.

## Business team

The business team is ultimately responsible for the business policies, which are enforced through the business decisions. The business team provides the input to the behavior of the business decisions. Depending on the level of adoption, the business team’s interaction varies.

The business team provides input to the rule development team for business policy changes to implement in the decisions. In one scenario, the business team can use Business Console to search, view, and modify specific rules from the existing system, requesting assistance from the rules development team if it is required.

In another scenario, the business team can view the rules from existing decisions in Decision Center and make recommendations to the rule development team to update specific rules that must be made to implement new business policies. In this mode, the business team has read-only access to the rulesets that contain the applicable rules.

In a third scenario, the business team updates the rules within a decision directly to implement changes that are required as business policies are updated. The business team tests the changes to the decisions by using the testing and simulation capabilities that are accessed through Decision Center consoles. The business team then notifies the systems administrator that a new version of a decision is available and must be deployed.

## 4.3 Sharing decision artifacts between z/OS and a distributed environment

Decision artifacts can be shared between distributed and z/OS environments if they are developed with this compatibility in mind.

When considering sharing decisions between z/OS and other platforms, ensure that the correct decision artifacts are available on each platform. As part of the decision lifecycle, the required artifacts can come from separate management systems.

Figure 4-6 shows the artifacts that are involved in a decision and the artifacts that are required to be deployed to various platforms.

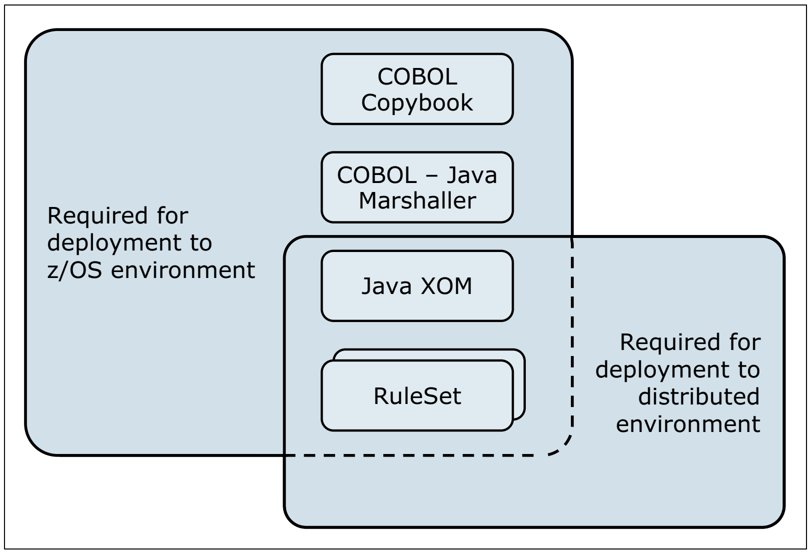

COBOL Copybook
COBOL – Java Marshaller
Required for deployment to z/OS environment
Java XOM
RuleSet
Required for deployment to distributed environment

*Figure 4-6 Deployment of decision artifacts*

**Marshaller:** The marshaller that is shown in Figure 4-6 is used to convert the COBOL or PL/I data to Java so that it can be accessed by the Java based ruleset. It is only required on the z/OS platform and must not be changed.

Normally, only the rulesets and the Java XOM are required on the distributed platform. Often in this configuration, the client is a Java application and is local to the Rule Execution Server for distributed. The Java XOM is part of the client application class path, so it is not explicitly deployed to the server as a resource. The rule session inherits the class path of the client application.

On zRES, the COBOL copybook or PL/I include file that was either imported into, or generated by, Rule Designer is required by the client COBOL application. The COBOL copybook ensures that the data layout is exactly the layout that is expected by the marshalling code. Because the client in this case is COBOL and the connection to the server is managed by the zRES API stub, the Java XOM resource must be deployed to the server. This deployment can be done either from the Rule Designer, Decision Center Business Console, or locally using supplied scripts.

In both cases, the Rule Execution Server requires that the ruleset is deployed within a RuleApp. This deployment is from either Rule Designer, Decision Center, or locally by using scripts.

When sharing the decision across multiple platforms, it is important to make sure that the decision lifecycle updates and deploys the correct parts of the decision when changes are made. If the underlying data structure definitions are not changed when updating a decision, only the rulesets within a RuleApp must be redeployed to make the new decision version available. If changes are made to the copybook that is used to create the BOM, or if the Java XOM was used to create the copybook, you need to redeploy artifacts. Redeploy artifacts on all platforms where the decision is implemented to minimize the chances of unexpected behavior or failures. For this reason, changes to the data model must be minimized after the decisions are in production.

## 4.4 Installation topologies for Decision Center

The locations of the Decision Center repository and the Decision Center Enterprise and Business consoles are largely independent of where the decision executes. Figure 4-7 on page 90 shows possible options for installing a Decision Center to be used with zRES. However, it is likely that more than one Decision Center console will be employed, possibly in different locations.

## 4.4.1 Basic topologies

Figure 4-7 shows three topologies using only one instance of the Decision Center console and the Decision Center repository.

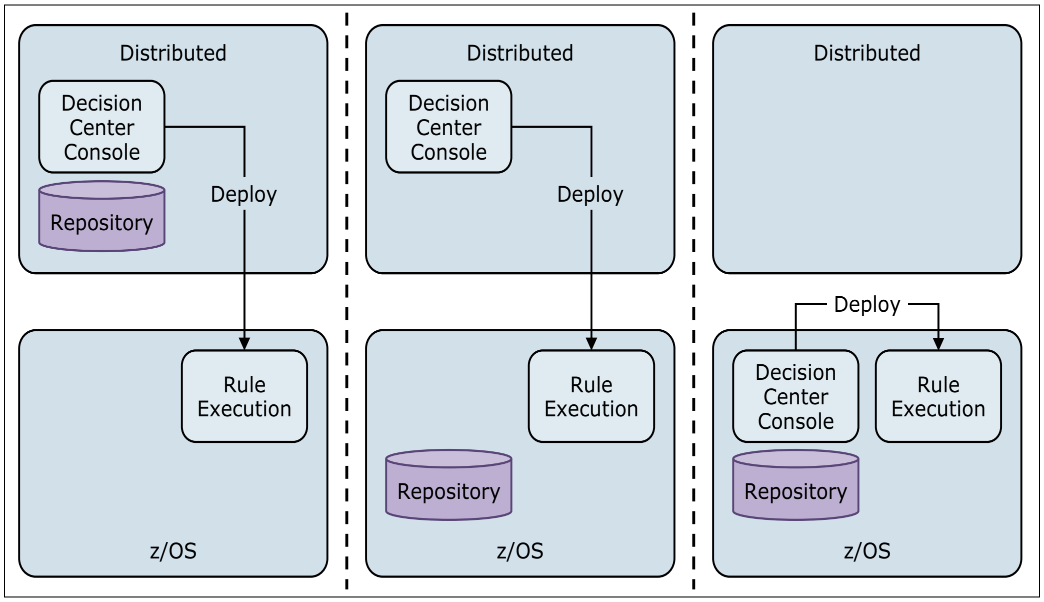


*Figure 4-7 Deployment options for Decision Center repository and console*

The Decision Center consoles require deployment to a web container. This web container can be any one of the supported Java Platform, Enterprise Edition (Java EE) application servers for distributed or WebSphere Application Server for z/OS. The Decision Center consoles also require Java Database Connectivity (JDBC) access to the repository database.

The following sections provide a brief description of the topologies that are shown in Figure 4-7.

### Topology 1: Decision Center console and repository on distributed

In this topology, both the Decision Center console and repository are hosted on a distributed or Linux for z Systems platform. The standard deployment from the Decision Center console to zRule Execution Server for z/OS (zRES) is by HTTP-based communication. As long as access is granted to the specific ports that are configured for deployment on the zRES or other Rule Execution Server instances, it is straightforward to deploy to any zRES instance. To Decision Center, the deployment interface to zRES looks the same as other Java EE deployed Rule Execution Server instances.

All security to the console is role-based using the underlying application server to perform authentication checks. You might prefer this configuration, because it does not require all users of the Decision Center console to be defined to z/OS security.

### Topology 2: Decision Center console on distributed and repository on z/OS

This topology is similar to Topology 1. However, the database is on a z/OS logical partition (LPAR). In this case, the Decision Center console must also have ports that are enabled in any firewall to allow remote client access to the database on z/OS. Generally, Decision Center is used less than the Decision Server, so the remote location of the database is not a performance problem.

You might prefer this configuration if the rule repository is required to have z/OS quality of service (QoS) associated with it or if the rule repository is managed on the same platform as the COBOL or PL/I source code repository. Generally, in this case, the security access to the remote database is delegated to an application server-level connection. That way, each Decision Center console user does not have to be defined to z/OS security.

#### Topology 3: Decision Center console and repository on z/OS

This topology places both the Decision Center console and repository on z/OS. The Decision Center console requires a WebSphere Application Server for z/OS instance in which to run and the repository requires an IBM DB2 instance. This topology can also be deployed to distributed Rule Execution Server instances and to zRES instances.

You might prefer this configuration if all administration and deployment of a project are contained within the z/OS teams.

## 4.5 Managing artifacts through the lifecycle

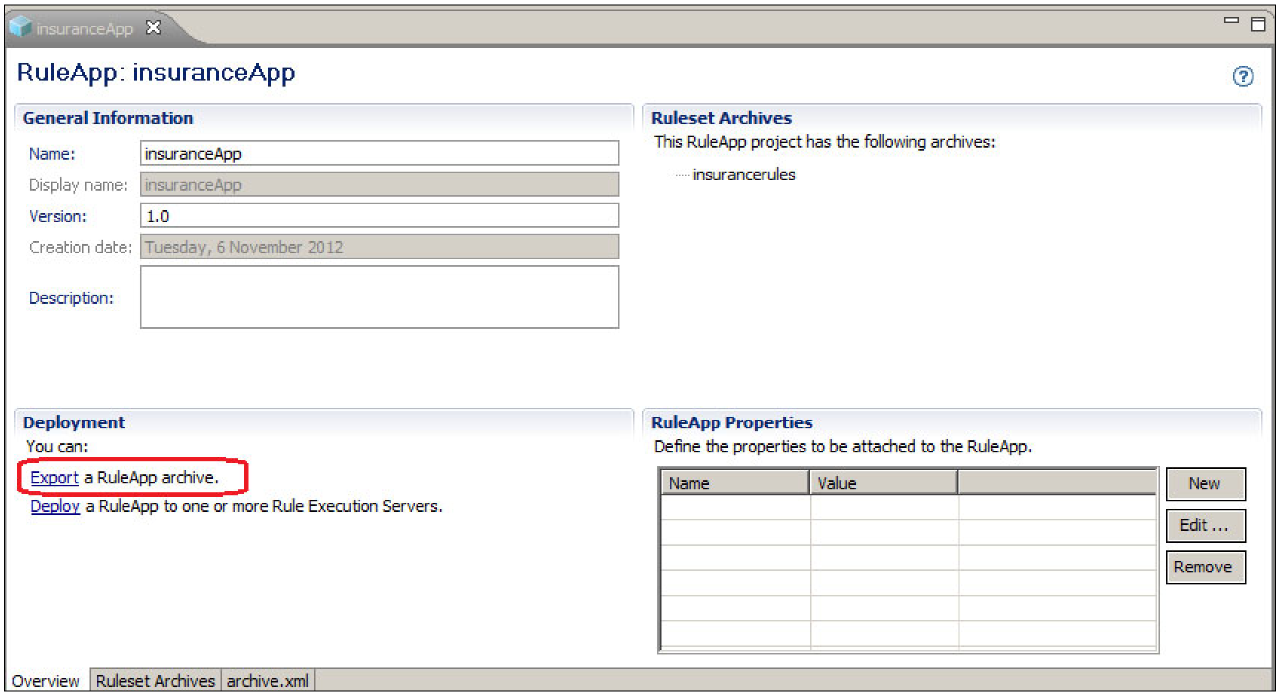
This section describes how the tools that are available in Operational Decision Manager for z/OS can be used to manage artifacts through the lifecycle.

### 4.5.1 Rule Designer

The Rule Designer is primarily a tool for creating rule artifacts. It is used for the creation of rules, ruleflows, decision trees, and other rules artifacts. It can be used to deploy rules to the zRES.

Because this tool is primarily aimed at an IT department rather than a system programmer, it is unlikely that it will be used to deploy a rule artifact to the production system. The main communication for the Rule Designer is with development and the test Rule Execution Servers, and with Decision Center repositories in these areas. It can, however, be used to export rules artifacts for later deployment to a rule server by using external scripts that implement the Representational State Transfer application programming interface (REST API).

Figure 4-8 shows the export window for a RuleApp.



insuranceApp

RuleApp: insuranceApp

**General Information**

| | |
|---|---|
| Name: | insuranceApp |
| Display name: | insuranceApp |
| Version: | 1.0 |
| Creation date: | Tuesday, 6 November 2012 |
| Description: | |

**Ruleset Archives**

This RuleApp project has the following archives:

insurancerules

**Deployment**

You can:

Export a RuleApp archive.

Deploy a RuleApp to one or more Rule Execution Servers.

**RuleApp Properties**

Define the properties to be attached to the RuleApp.

| Name | Value | |
|---|---|---|
| | | |

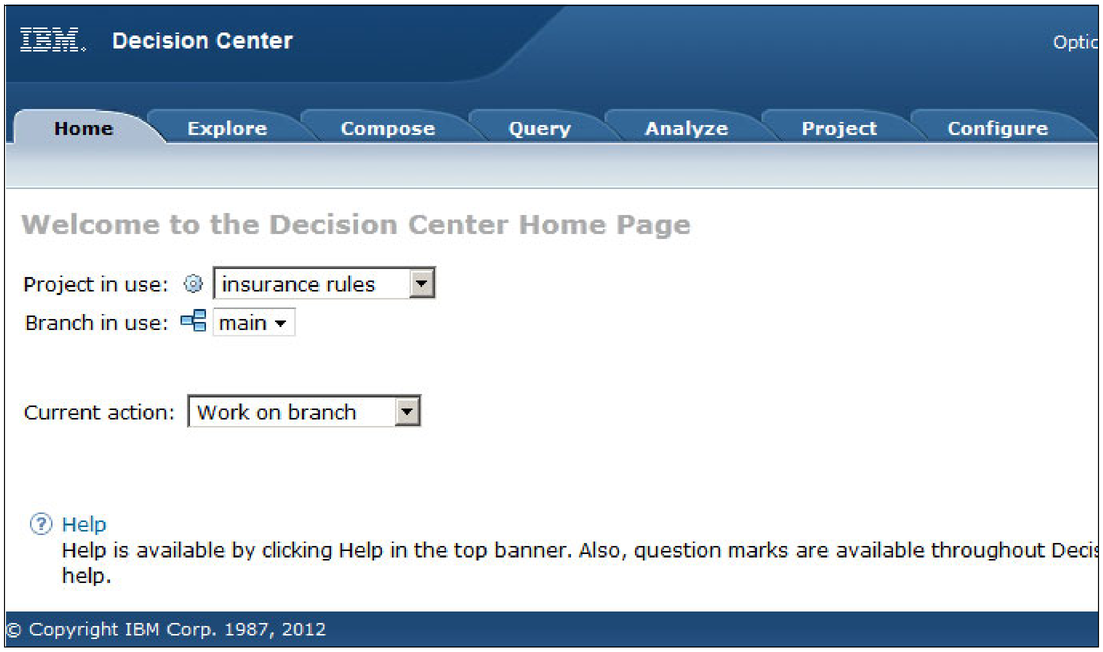
*Figure 4-8 Deployment and export from Rules Designer*

Anything that is deployed from the Rule Designer to a zRES takes effect immediately.

The Rule Designer can also be used to publish rules and the XOM to a Decision Center repository by first connecting to the appropriate Decision Center and then synchronizing with it. This allows for publishing rules to the main root or to one of the branches. Roots and branches are described in “Versioning” on page 93.

## 4.5.2 Decision Center

The Decision Center, as shown in Figure 4-9, can be configured to support a variety of users, allowing it to be a central tool in managing artifacts currently in use. Anything deployed from the Decision Center will be effective immediately on the zRES to which it has been deployed.



Welcome to the Decision Center Home Page

Project in use: insurance rules

Branch in use: main

Current action: Work on branch

Help

Help is available by clicking Help in the top banner. Also, question marks are available throughout Decis
help.

© Copyright IBM Corp. 1987, 2012

*Figure 4-9 Decision Center on startup*

When configured for business users, it might allow the modification of specific rules and events and viewing of more complex scenarios. Deploying might not be appropriate in these circumstances. Rules can be updated or modified. Testing a specific rule can also be done.

When used by a developer, rule artifacts can be created and modified, and then deployed to a zRES. Although, more often, the Rule Designer is used for the main development and Decision Center is used as a maintenance tool.

When used by a system programmer, it is likely that the majority of functions are enabled. Although in some cases, it might be useful to disable the editing capabilities on artifacts that have been developed by the other teams. The system programmer's primary use of the Decision Center is to deploy the rule artifacts to the appropriate Rule Execution Servers. The Decision Center might also be used for exporting so that rules can be deployed to the production server by using the REST APIs, as described in 4.5.4, "REST API" on page 94.

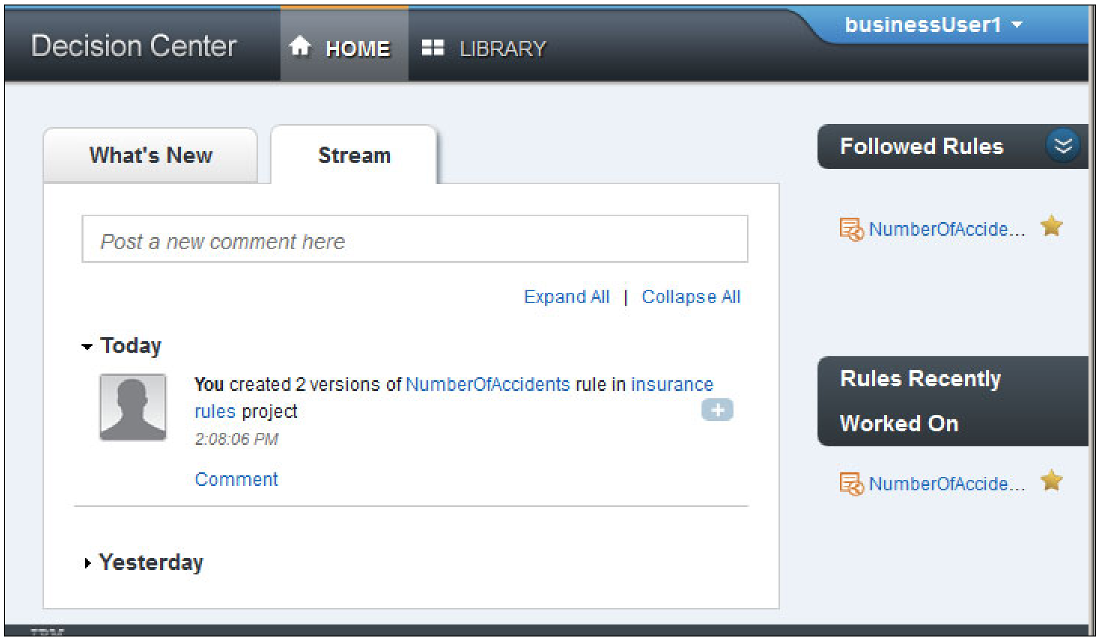
### Versioning

When a new version of an element is modified, the Decision Center creates an archived version of that element. Therefore, the history of a particular element can be tracked by reviewing those archived versions.

## 4.5.3 Business Console

The Business Console, as shown in Figure 4-10, is primarily a tool for use by business users. It allows for the searching, viewing, editing, test, simulation, and deployment of rules. It also allows timelines to be used to see the history of a rule.

The editing of rules in the Business Console causes an archived version of the rules to be created as described in “Versioning” on page 93.



You created 2 versions of NumberOfAccidents rule in insurance rules project
2:08:06 PM

*Figure 4-10 Business Center*

## 4.5.4 REST API

Operational Decision Manager allows for the deployment of rules and supporting artifacts by using the REST API services. Therefore, the deployment of a rule artifact can be scripted to ensure that it is deployed in the same way each time. It also means that rule artifacts can be published without using an HTTP connection between the tool and zRES. This is a useful method when deploying to a production environment because it preserves security on the production system.

As shown in Figure 4-11 on page 95, to use this method, first the rule artifact is exported, and then, it is deployed to the zRES server by using scripts by using the REST API. The archive file can be copied to the appropriate location, which means that HTTP connections are not involved.

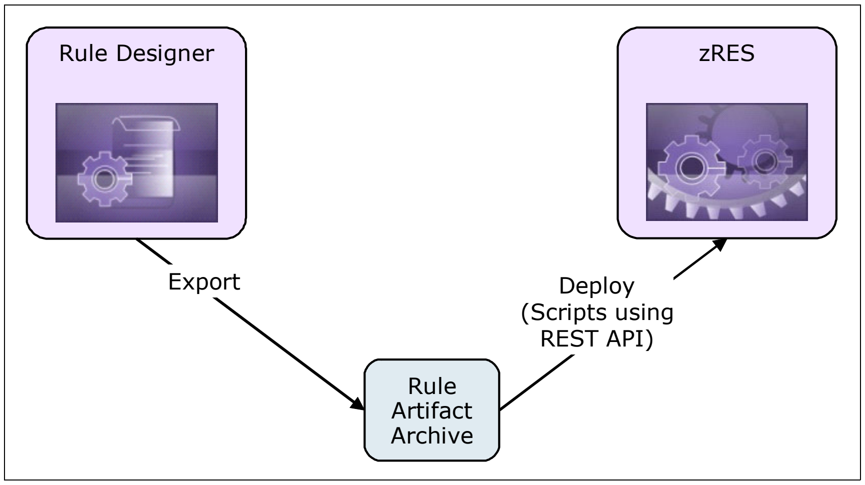


*Figure 4-11 Deploying a rule artifact by using REST API services without using HTTP*

### 4.5.5 Deployment scripts

By using JCL, is it possible to script the deployment of rules and the associated artifacts. It is possible to deploy directly to the persistence layer using HBRDPLOY. However, this only affects the database and does not inform running applications (such as the zRES) that changes have occurred. Therefore, any changes that use this method need the zRES to be restarted for the changes to take effect.

The preferred mechanism is to use the HBRDPLYC, which uses the REST API to deploy directly to a running RES Console. This notifies the zRule Execution Servers to update their copy of the rules.

## 4.6 Usage of defined rules

After a rule artifact is defined to the Decision Server, it is available for use by calling applications. If the artifacts are deployed using the REST API or the RES Console, no restart of the Decision Server is required.

The rule can then be modified separately to the application by using the methodologies described in this chapter. The application using the rule does not require modification for an update to be made to that rule. By using a deployment method that requires no restart, there does not need to be any interruption of service for the rules to be updated.

5

# Invoking Operational Decision Manager from COBOL and PL/I clients

This chapter describes the design of the decision interface. It describes the mapping between the COBOL and PL/I data structures passed into the decision run time and the Java structures that are used to execute the rules. This chapter explains how you can then customize this mapping.

This chapter also describes starting a new project for deployment from a COBOL copybook. The same steps also apply for PL/I data structures.

The following topics are covered in this chapter:

## 5.1 Designing the decision interface

The starting point for any successful decision project is to design the decision interface correctly. Often, there is an existing copybook that is used by the application that is being enabled to access Operational Decision Manager. The natural starting point seems to be to reuse this data structure as the interface to the decision. Although this approach might seem to be the easiest, there are benefits to be gained by giving the interface more consideration.

When designing the decision interface, consider why you are externalizing business rules into an external decision server. One of the main reasons might be that the business decisions change on a shorter lifecycle than the applications that invoke them. Therefore, you might want to isolate the application from changes to the business rules in the decision. However, you also might want to isolate the business decision from maintenance changes that occur in the application.

Changes in the application can alter the copybook that the application uses. Application changes result in the need to import the copybook again to update the business object model (BOM) that is used in the rules. These changes can potentially disrupt rules that are already authored.

Also, the data in the copybook might not be suitable for use with rule authoring. Application copybooks in COBOL and PL/I are often a mix between a representation of the business data and its specific entries. For example, often COBOL FILLER statements are put in to align data, or fields are created to hold return codes or other diagnostic information. These fields have no business relevance to the decision and are confusing if they are included in the BOM.

It is important to ensure that the data passed across to the business decision contains all the required information to successfully make the decision. When thinking about the data, consider information that might not be used in the business rules that is embedded in the application today. This information might be useful to develop a better business decision after it is externalized. It is far easier to pass more data across when designing the decision interface from the outset than to re-engineer the interface later or to add code into the decision to retrieve external data.

It is a preferred practice to design the interface to a business decision in the same way that you design a service interface. Consider the following information:

- The data that is used in the decision today
- The data that is easily available to the application
- The data that is required today, and potentially in the future, to maintain the decision after it is externalized

Create a copybook or include file to hold that information specifically for the decision interface that can be versioned for that purpose. The additional cost of a few COBOL MOVE or PL/I assign statements is outweighed by the flexibility that this approach provides in isolating changes in both the decision and the calling application.

## 5.2 Coding the COBOL or PL/I client application

To access the zRule Execution Server for z/OS (zRES), the application must use the supplied client API. This API consists of the following API calls:

| | |
|---|---|
| **HBRCONN** | To connect to the server |
| **HBRRULE** | To execute a decision |
| **HBRDISC** | To disconnect from the server |

Each API call takes the HBRWS copybook-defined structure as a parameter. These calls are shown in 3.2.10, “Building a COBOL application for rule execution” on page 64.

## 5.2.1 HBRWS header structure

The HBRWS header structure is required for all zRES client API calls. For COBOL, it is in the `++HBRHLQ++.SHBRCOBS` data set. For PL/I, it is in the `++HBRHLQ++.SHBRPLIS` data set. It must be included in any client programs that call the zRule Execution Server for z/OS.

Example 5-1 shows the layout of the HBRWS header structure for COBOL.

*Example 5-1 Layout of the HBRWS header structure*

```
01 HBRA-CONN-AREA.
    10 HBRA-CONN-EYE                PIC X(4)   VALUE 'HBRC'.
    10 HBRA-CONN-LENTH              PIC S9(8)  COMP  VALUE +3536.
    10 HBRA-CONN-VERSION            PIC S9(8)  COMP  VALUE +2.
    10 HBRA-CONN-RETURN-CODES.
       15 HBRA-CONN-COMPLETION-CODE PIC S9(8)  COMP  VALUE -1.
       15 HBRA-CONN-REASON-CODE     PIC S9(8)  COMP  VALUE -1.
    10 HBRA-CONN-FLAGS              PIC S9(8)  COMP  VALUE +1.
    10 HBRA-CONN-INSTANCE.
       15 HBRA-CONN-PRODCODE        PIC X(4)   VALUE SPACES.
       15 HBRA-CONN-INSTCODE        PIC X(12)  VALUE SPACES.
       15 HBRA-CONN-SSID            PIC X(4)   VALUE SPACES.
       15 HBRA-CONN-RESERVED PIC X(4)   VALUE SPACES.
    10 HBRA-RESERVED01              PIC S9(8)  COMP  VALUE 0.
    10 HBRA-RESERVED02              PIC S9(8)  COMP  VALUE 0.
    10 HBRA-RESERVED03              PIC S9(8)  COMP  VALUE 0.
    10 HBRA-CONN-RULE-CCSID         PIC S9(8)  COMP  VALUE 0.
    10 HBRA-CONN-RULEAPP-PATH       PIC X(256) VALUE SPACES.
    10 HBRA-RESPONSE-AREA                     VALUE SPACES.
       15 HBRA-RESPONSE-MESSAGE     PIC X(1024).
    10 HBRA-RA-INIT                           VALUE LOW-VALUES.
       15 HBRA-RESERVED04           PIC X(1792).
    10 HBRA-RA-PARMETERS
       REDEFINES HBRA-RA-INIT.
       15 HBRA-RA-PARMS  OCCURS 32.
          20 HBRA-RA-PARAMETER-NAME     PIC X(48).
          20 HBRA-RA-DATA-ADDRESS       USAGE POINTER.
          20 HBRA-RA-DATA-LENGTH        PIC 9(8) BINARY.
    10 HBRA-RESERVED.
      15 HBRA-RESERVED05            PIC X(12).
      15 HBRA-RESERVED06            PIC X(64).
      15 HBRA-RESERVED07            PIC X(64).
      15 HBRA-RESERVED08            PIC X(128).
      15 HBRA-RESERVED09            PIC X(132).
```

The entire HBRA-CONN-AREA must be passed on each of the three API calls to zRule Execution Server for z/OS. The following sections describe the important elements of this structure.

## HBRA-CONN-RETURN-CODES

The HBRA-CONN-RETURN-CODES element provides completion and reason codes to the requested API call. If these responses do not have a zero value on their return from an API, the user documentation provides more details about the error that occurred.

## HBRA-CONN-RULEAPP-PATH

The HBRA-CONN-RULEAPP-PATH element is important for the data structure. This value is used by zRule Execution Server for z/OS to identify which decision to execute on the specific data. After a decision is deployed to zRule Execution Server for z/OS, the value for the RULEAPP-PATH can be identified by logging on to the RES console and viewing the deployed decision.

The RULEAPP-PATH uses this structure:

*/<RULEAPP-NAME>/<RULEAPP-VER>/<RULESET-NAME>/<RULESET-VER>*

The *RULE-APP-VER* and *RULESET-VER* are generally required only if the client wants to execute a particular version of a decision that is deployed to zRule Execution Server for z/OS. In most cases, this is the latest version of the decision. In this case, the path is simplified to this structure:

*/<RULEAPP-NAME>/<RULESET-NAME>*

For example, the Miniloan example that is supplied with Operational Decision Manager uses this structure:

*/MiniLoanDemoRuleApp/MiniLoanDemo*

The different combinations and what they mean can be found in Example 5-2. A practice used for some customers is that when business rules change, they increment the ruleset version and when the data structure (for example, COBOL copybook or PL/I include) changes, they increment the rule application version.

*Example 5-2 Using /MiniLoanDemoRuleApp/MiniLoanDemo as the Rule Application and Ruleset, the following are the different combinations that can be used for calling applications and what each does*

```
Scenario 1 to always call the latest RuleApp and Ruleset Version use:
/MiniLoanDemoRuleApp/MiniLoanDemo
Scenario 2 to always call a specific RuleApp version, but latest Ruleset Version
use: /MiniLoanDemoRuleApp/1.0/MiniLoanDemo
Scenario 3 to always call the latest RuleApp version, but specific Ruleset Version
use: /MiniLoanDemoRuleApp/MiniLoanDemo/1.0
Scenario 4 to always call a specific RuleApp version and a specific Ruleset
Version use: /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0
```

## HBRA-RESPONSE-AREA

If any text messages or warnings are returned by the Java portion of the server, they are returned in the HBRA-RESPONSE-AREA data area to help you diagnose any problems. A preferred practice is to write this area out to an application log if a nonzero return code is received.

## HBRA-RA-PARAMETERS

The HBRA-RA-PARAMETERS section is required only on the HBRRULE API call. The HBRA-RA-PARAMETERS section provides the user data that is used to evaluate the decision.

The structure is a list of triplets to pass the parameter name, a pointer to the data in working storage, and the length of the data:

- HBRA-RA-PARAMETER-NAME is the name of the parameter as it is known to the decision. This parameter name is defined as part of the rule-authoring process. The parameter name is case-sensitive and must be added as a character string, exactly as it is defined in the ruleset.
- HBRA-RA-DATA-ADDRESS is a pointer to the location of the parameter in your working storage. Normally, this element points to the address of the 01-level element in your copybook that you imported to define this parameter to the decision. It is normally set like the following example:

  ```
  set HBRA-DATA-ADDRESS(1) to address of Borrower
  ```
- HBRA-RA-DATA-LENGTH defines the length of the data structure to which the HBRA-DATA-ADDRESS points. Ensure that you set this element correctly so that all necessary data is passed across to the decision execution. You can use COBOL or PL/I intrinsic functions to calculate this value, for example:

  ```
  move LENGTH OF Borrower to HBRA-DATA-LENGTH(1)

  or

  HBRA_RA_DATA_LENGTH(1) = SIZE (BORROWER);
  ```

## 5.2.2 HBRCONN API call

You use the HBRCONN API call to establish a connection to the zRule Execution Server for z/OS server or for the Rule Execution Server for WebSphere Application Server on z/OS. The HBRA-CONN-AREA data structure is passed as a parameter to this call.

In CICS, when using a Rule Execution Server for z/OS stand-alone server, Rule Execution Server on WebSphere Application Server for z/OS, or the locally optimized Java virtual machine (JVM) server deployment, the connection call is made at start or when the HBRC transaction is run and not during the HBRCONN call.

**Important:** The HBRCONN API call should still be included in the CICS client applications as it is used to perform transaction-based initialization.

## 5.2.3 HBRRULE API call

The HBRRULE API call invokes the decision for evaluation. The HBRA-CONN-AREA data structure is passed as a parameter and must contain references to the ruleset parameter data that is required for evaluating the decision.

Multiple HBRRULE calls should be made within a single HBRCONN/HBRDISC pair. This is also the preferred practice when performance is a consideration as HBRCONN and HBRDISC have a processing overhead associated with them.

## 5.2.4 HBRDISC API call

You use the HBRDISC API call to disconnect from the server after all decisions are evaluated for this application. The HBRA-CONN-AREA is passed as a parameter.

In CICS, HBRDISC does not disconnect the CICS region explicitly. Instead, for CICS, this is achieved by using the HBRD transaction.

**Important:** The HBRDISC API call should still be included in the CICS client applications as it is used to perform transaction-based cleanup.

## 5.3 Mapping from the COBOL copybook

This section describes the following COBOL topics:

- Structure of a COBOL-based rule project
- Supported COBOL and PL/I data types
- Creating custom converters
- Mapping level-88 constructs into BOM domain types

This section focuses on COBOL, however the general concepts also apply to PL/I. Where PL/I is different, it is highlighted in the text.

### 5.3.1 Structure of a COBOL-based rule project

In Chapter 3, “Getting started with business rules” on page 23, when importing a COBOL copybook, a Java project is generated which is the Java Execution Module (XOM) project.

#### XOM project

The XOM project contains a Java representation of the COBOL or PL/I data structures. Each level-01 item in the copybook, both group and elementary items, is mapped to a Java class. Each non-level-01 group item is also mapped to a class.

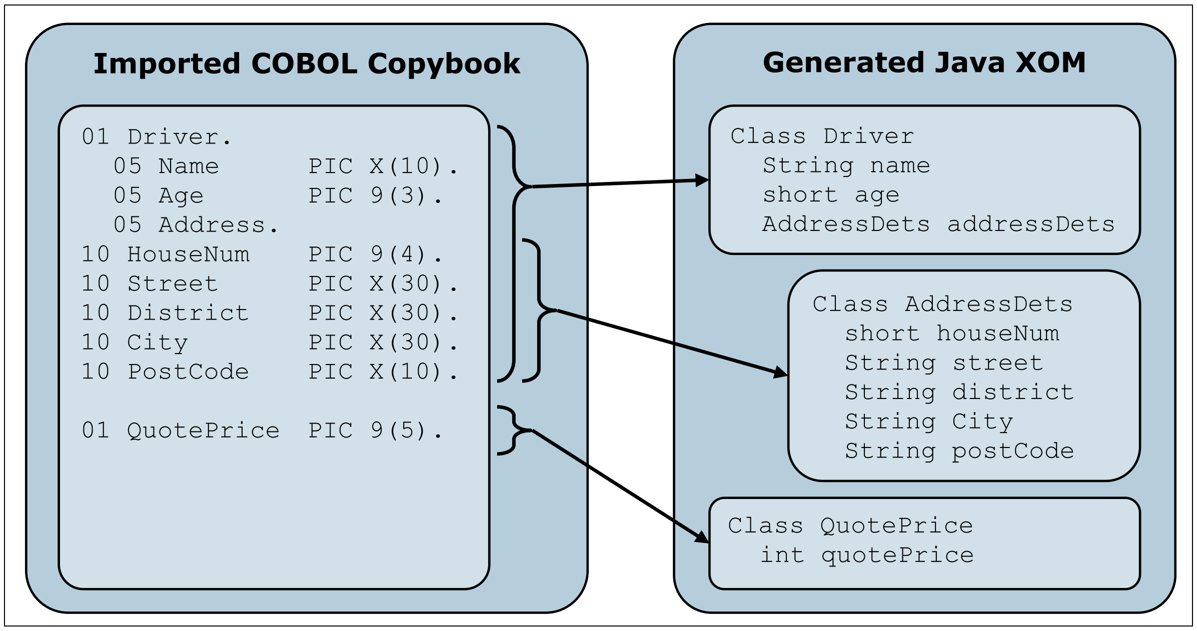


*Figure 5-1 Imported copybook level-01 items are mapped to Java classes*

**Important:** Only level-01 items can be used as ruleset parameters.

### Marshalling

During the processing of requests from a COBOL or PL/I application, the zRule Execution Server for z/OS first converts the native data into Java XOM objects. After executing the ruleset, the zRule Execution Server for z/OS then converts the Java objects back to native data.

The marshaller classes are not externalized and cannot be altered. The marshaller is produced by Rule Designer during the importing of the COBOL copybook or PL/I include file.

## 5.3.2 Supported COBOL and PL/I data types

This section describes the supported and unsupported COBOL data types.

### Supported and Unsupported types

The supported data types can be in the Operational Decision Manager (ODM) IBM Knowledge Center:

COBOL

http://www-01.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.dserver.rules.designer.author/cobol_topics/con_xomguidelines_cobol2java.html

PL/I

http://www-01.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.dserver.rules.designer.author/pl1_topics/con_pl1_xomguidelines_pl12java.html

Some data types are not supported because there are no suitable Java types to which to map. Do not include these types in the import copybook.

If the copybook structure cannot be changed, for example, for compatibility with existing applications, consider this work-around. In the copybook for importing, change the data item to a corresponding ordinary alphanumeric or national type of the same length. Then, these data items are mapped to Java strings in the generated XOM class.

There are also two unsupported Occurs Depending On (ODO) table situations:

- ODO table within a fixed-length table
- ODO table sharing the ODO object

Consider using a fixed-length table to work around these limitations.

### Converter

You can use type converters to change the Java type to which a COBOL or PL/I data item is mapped. There are two built-in converters:

- String to boolean converter
- String to Date converter

You can also implement custom converters or set a XOM field to a custom-defined domain class.

## 5.3.3 Creating custom converters

In Chapter 3, “Getting started with business rules” on page 23, you learned about how to use the built-in type converters to map a COBOL data item to Java Boolean or Date type. There are cases when the built-in converter cannot meet your needs. You can then write a custom converter.

Consider this example. In a COBOL program, instead of using only `T` to indicate true, the program also accepts `t`, `Y`, and `y` as true values. But the built-in Boolean converter can accept only one character as the true value. So, in this case, you need to write a custom converter.

A *custom converter* is a normal Java class. The class must be annotated with the TypeConverter annotation. In the Converter dialog, users can select those classes with only the TypeConverter annotation, along with the built-in converters.

You then need to implement an init method. This method is called by the run time to initialize the converter with user-defined properties. There is also an optional TypeConverterProperties annotation with which you can define the list of expected property keys. In this example, two keys are defined:

| | |
|---|---|
| **true-values** | A comma-separated list of characters that indicate true |
| **false-value** | The default value that indicates false |

In the init method, the values of these properties are retrieved. Then, the list of true values to the trueValues field is saved and the falseValue field is set to the false-value character. See Example 5-3.

*Example 5-3 Custom converter code (part 1 of 2)*

```
@TypeConverter
@TypeConverterProperties({ "true-values", "false-value" })
public class MyBooleanConverter {
   private List<String> trueValues;
   private String falseValue;

   public void init(Map<String, String> props) {
      String trueStr = props.get("true-values");
      trueValues = Arrays.asList(trueStr.split(","));
      falseValue = props.get("false-value");
   }
```

Now, you can define two conversion methods:

```
public <TargetType> convertToTarget(<SourceType> value)
public <SourceType> convertToSource(<TargetType> value) {
```

The <*SourceType*> is the default Java type that is directly mapped from COBOL data and the <*TargetType*> is the type that you want to use in the generated XOM. The convertToTarget is called during unmarshalling, and the convertToSource is called during marshalling.

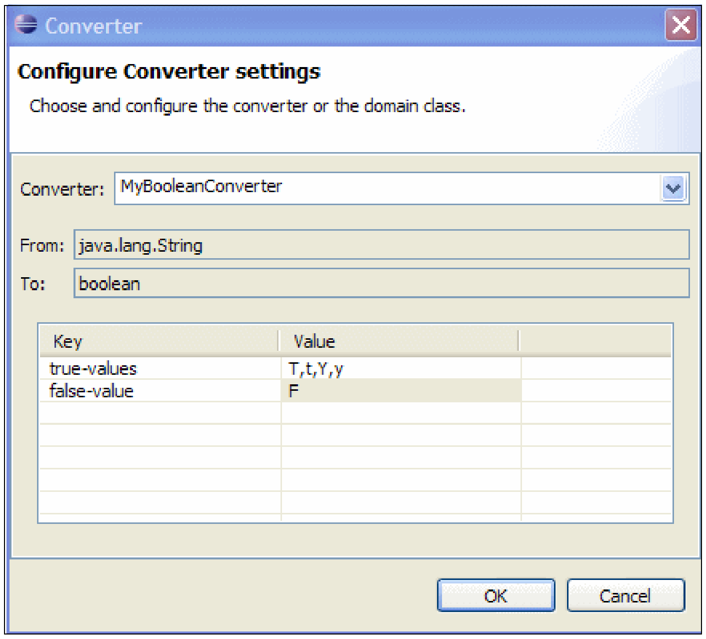
In Example 5-4 on page 105, if the value that comes from COBOL is within the trueValues, the result is true. Any other values convert to false. During marshalling, if the Java value is true, the first of the possible true values is used, which is `T` in this case. If the Boolean value is false, the false value `F` is used.

*Example 5-4 Custom converter code (part 2 of 2)*

```
   public synchronized boolean convertToTarget(String value) {
      return trueValues.contains(value);
   }
   public synchronized String convertToSource(boolean value) {
      return value ? trueValues.get(0) : falseValue;
   }
}
```

After the custom converter is implemented, you must add the Java project to the XOM path of the rule project. Then, when adding a COBOL XOM, in the Converter dialog, you can choose the converter that you defined and set the property values (see Figure 5-2).

In the generated marshaller code, the user-provided properties in this dialog are sent as parameters in the call of the init method. Therefore, the converter is initialized before any conversion operation.


Converter
Configure Converter settings
Choose and configure the converter or the domain class.
OK
Cancel

Converter: MyBooleanConverter

From: java.lang.String

To: boolean

| Key | Value |
|---|---|
| true-values | T,t,Y,y |
| false-value | F |

*Figure 5-2 Converter dialog with custom converter*

## 5.3.4 Mapping level-88 constructs into BOM domain types

A *domain* can restrict the possible values that a type element in BOM can accept. During rule authoring, the editor suggests values according to the enumerated domains. A semantic check is also performed to check that the business rule does not use a value outside the defined domain.

The copybook that is shown in Example 5-5 was used in Chapter 3, “Getting started with business rules” on page 23.

*Example 5-5 COBOL copybook*

```
              05  VEHICLE.
               10  VEC-ID                  PIC  X(15).
               10  MAKE                    PIC  X(20).
               10  MODEL                   PIC  X(20).
               10  VEC-VALUE               USAGE COMP-1.
               10  VEC-TYPE                PIC  X(2).
                   88 SUV     VALUE 'SU'.
                   88 SEDAN   VALUE 'SD'.
                   88 PICKUP  VALUE 'PU'.
```

The vehicle type VEC-TYPE data item has three level-88 items, which define the valid values for this item: `SU`, `SD`, and `PU`. When importing this copybook into the Rule Designer, the level-88 items are mapped to methods. These methods are helpful to check the value of the field or to assign the correct value to the field, but they cannot prevent the field from being assigned incorrect values. Ideally, a user might use valid values only with the vehicle type, or the user can use actual vehicle types instead of codes to represent the vehicle types. A domain must be defined to meet this requirement. In the COBOL XOM import wizard, a domain converter to map the vehicle type to a Java domain type needs to be defined.

Follow these steps:

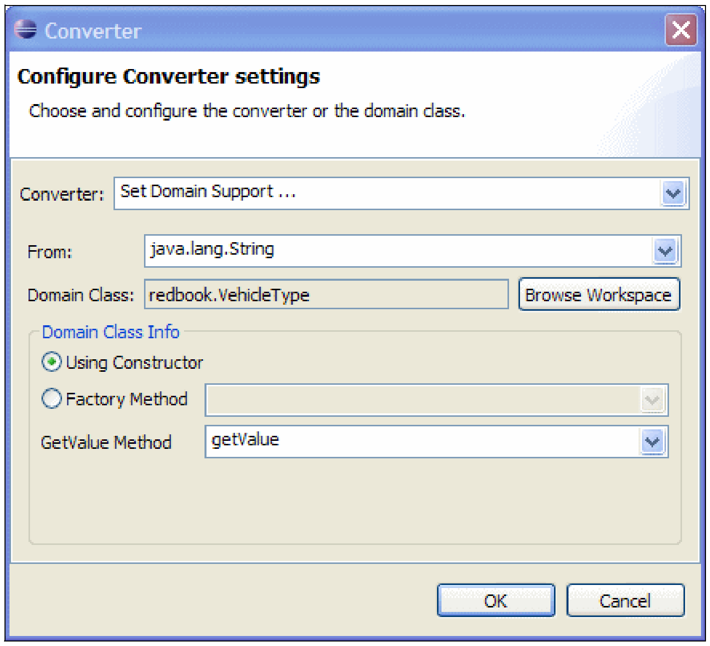
1. Implement a Java class as the XOM for the domain definition. Example 5-6 is the sample code. In this class, define a constructor with a string parameter. This constructor accepts the vehicle type codes and creates a VehicleType object. You must also implement a getValue method to retrieve the string code from the VehicleType object.

   *Example 5-6 Java class for domain vehicle type*

   ```
   package redbook;
   public class VehicleType {
     public final static VehicleType SUV = new VehicleType("SU");
     public final static VehicleType SEDAN = new VehicleType("SD");
     public final static VehicleType PICKUP = new VehicleType("PU");
     private String code;
     public VehicleType(String code) {
       this.code = code;
     }
     public String getValue() {
       return code;
     }
   }
   ```

2. In Rule Designer, add the project to the Java XOM path of the rule project.
3. When you import a COBOL copybook, define a converter for the vehicle-type item. In the dialog that is shown in Figure 5-3 on page 107, when you select **Set Domain Support** for the Converter field, you can select the From type. The From type is the Java type when the COBOL data is first unmarshalled and before the converter is applied.

   You need to provide the domain class name that you want to use. You can then provide detailed information for the domain class. Select **Using Constructor** to convert the string codes to the domain object and then select the **GetValue Method** to convert the domain object to string.


Converter

**Configure Converter settings**

Choose and configure the converter or the domain class.

Converter: Set Domain Support ...

From: java.lang.String

Domain Class: redbook.VehicleType

Domain Class Info

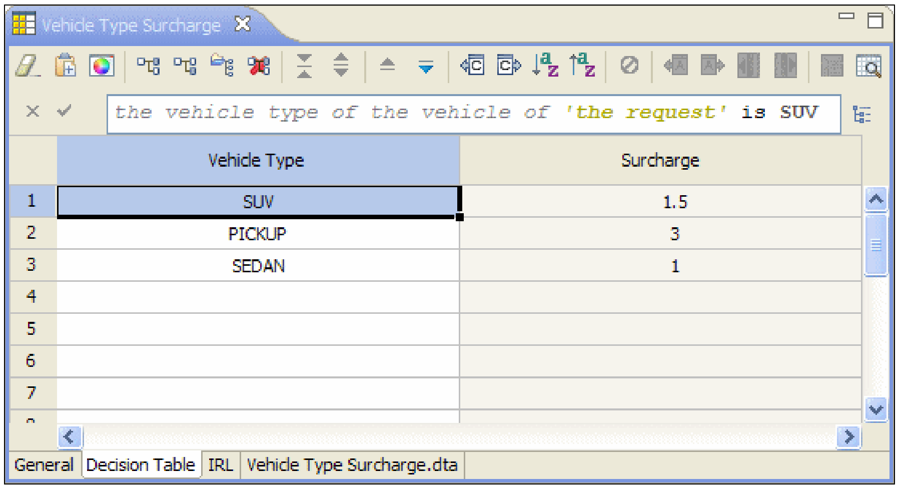
[✓ Using Constructor]

Factory Method

GetValue Method: getValue

*Figure 5-3 Converter dialog to set up the domain type*

4. You must create a BOM entry for the domain class. Then, create a BOM entry for the COBOL XOM. With the correct verbalization, you can use the domain in rule authoring (Figure 5-4).


Vehicle Type Surcharge

the vehicle type of the vehicle of 'the request' is SUV

|   | Vehicle Type | Surcharge |
|---|---|---|
| 1 | SUV | 1.5 |
| 2 | PICKUP | 3 |
| 3 | SEDAN | 1 |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

*Figure 5-4 A sample decision table that uses the COBOL domain*

5. In the converter dialog, you can also use a static factory method instead of a constructor to define a domain converter. Example 5-7 on page 108 is a sample implementation of a static factory method.

*Example 5-7 Static factory method in the domain class VehicleType*

```
public static VehicleType getVehicleType(String code) {
    if (SUV.code.equals(code))
      return SUV;
    else if (SEDAN.code.equals(code))
      return SEDAN;
    else if (PICKUP.code.equals(code))
   return PICKUP;
    else
   return null;
  }
```

## 5.4 Changing the client application to reach the rule server

The client application needs to be bound to the ODM stub code depending on the server that makes the business decision.

### 5.4.1 Batch application

For a batch application, the client application needs to be bound to the Operational Decision Manager stub:

```
//HBRLIB DD  DSN=++HBRHLQ++.SHBRLOAD
   INCLUDE HBRLIB (HBRBSTUB)
```

The job that runs the client application will also have the following line that tells the application which server to connect:

```
//HBRENVPR DD DISP=SHR,DSN=++HBRWORKDS++..SHBRPARM(HBRBATCH)
```

### 5.4.2 IMS application

For an IMS application, the client application needs to be bound with the Operational Decision Manager stub for IMS:

```
//HBRLIB DD DSN=++HBRHLQ++.SHBRLOAD
   INCLUDE HBRLIB (HBRISTUB)
```

More information about working with IMS is in Chapter 11, “Configuring IBM IMS to work with Operational Decision Manager” on page 169.

### 5.4.3 CICS application

For a CICS application, the client application needs to be bound to the Operational Decision Manager stub:

```
//HBRLIB DD DSN=++HBRHLQ++.SHBRCICS
   INCLUDE HBRLIB (HBRCSTUB)
```

More information about working with the CICS JVM is in Chapter 10, “Configuring IBM CICS to work with Operational Decision Manager” on page 153.

## 5.4.4 WebSphere Optimized Local Adapters batch application

For a WebSphere Optimized Local Adapters (WOLA) batch application, the client application is bound as described in 5.4.1, “Batch application” on page 108. However, in addition, the job that runs the application needs to include the WOLA parameter file:

```
//HBRENVPR DD DISP=SHR,DSN=++HBRWORKDS++..SHBRPARM(HBRWOLA)
```

More information about working with WOLA is in Chapter 13, “Configuring IBM WebSphere Optimized Local Adapters support” on page 183.

6

# Advanced topics for decision authoring

This chapter describes the authoring rules for deployment to z/OS. It does not explain general rule authoring. However, it does describe in more detail the mapping between the COBOL or PL/I data structures that are passed into the decision run time and the Java structures that are used to execute the rules. This chapter also explains how you can customize that mapping.

This chapter describes starting a new project for deployment from either a COBOL copybook, a PL/I include file, or an existing rule project based on Java. It then explains how you can extend the capabilities of the decision execution by adding custom methods into the business object model (BOM).

The following topics are covered in this chapter:

- 6.1, “Starting from an existing Java based BOM project” on page 112
- 6.2, “Extending the capability of the rule execution with BOM methods” on page 114
- 6.3, “Augmenting ruleset parameters from external data sources” on page 117
- 6.4, “Considerations for sharing rules between z/OS and distributed applications” on page 119
- 6.5, “Authoring considerations for performance” on page 121

## 6.1 Starting from an existing Java based BOM project

If rule projects are currently in production on distributed systems, and you want to migrate the rule application to the mainframe, you can enable the BOM for a COBOL or PL/I application and generate a COBOL copybook or PL/I include file.

**Important:** Only a BOM that originated from a Java execution object model (XOM) is supported. A BOM that originated from an XML-based dynamic XOM is not supported.

### 6.1.1 Mapping Java data structures to COBOL

This section explains mapping Java data structures to COBOL. However, the same principles are used when mapping from Java to PL/I.

#### Aggregation data structure

When mapping Java BOMs to COBOL, only BOM classes with aggregation relationships are supported. If there are object references, a simple hierarchical data structure is supported. Complex object graphs are not supported. For example, in the sample application that is used in this document, the Insurance class has a field called Vehicle of type Vehicle. The vehicle information is part of the insurance data, and it is a simple hierarchical data structure. This example is supported.

But, if the Vehicle class references Insurance either directly or indirectly through other classes, the data structure is not hierarchical. It contains loops. In this case, the BOM-to-COBOL mapping is not supported. In addition, class inheritance is not supported. The BOM must not include the following usage:

- Inheritance
- Loop reference, including self-reference
- Static attribute

Also, ensure that the BOM classes follow JavaBeans naming guidelines, such as well-formed getters and setters; otherwise, the generated marshaller classes contain incorrect code.

#### General mapping rule

A BOM class is mapped to a COBOL group. Fields of the basic Java type are mapped to child elementary data items. And, fields of the class type are mapped as subgroups (Example 6-1).

*Example 6-1 A BOM with two classes*

```
package xom;
public class Request {
    public xom.Driver primaryDriver;
    public xom.Driver secodaryDriver;
}
public class Driver {
public short age;
public string name;
}
```

When this BOM is mapped to a COBOL copybook, the primaryDriver and secondaryDriver fields are generated as two groups with the same structure (Example 6-2).

*Example 6-2 Copybook with two similar groups*

```
01 request.
          02 primaryDriver.
              03 age pic S9(5).
              03 name pic X(20) value SPACE.
          02 secodaryDriver.
              03 age pic S9(5).
              03 name pic X(20) value SPACE.
```

An array is mapped to a table, and a collection is mapped to a size data item and a table (Example 6-3).

*Example 6-3 A BOM with an array and list*

```
package xom;
public class Request {
    public xom.Driver[] drivers;
    public java.util.List vehicles domain 0,* class xom.Vehicle;
}
public class Driver {
public short age;
public string name;
}
public class Vehicle {
    public string vehicleId;
    public double vehicleValue;
}
```

The driver's field is an array, so the COBOL data item is a fixed-length table. The vehicles field is a list of Vehicle objects. So, in the generated copybook, vehicles-Num is used as the actual size of the table vehicles (Example 6-4).

*Example 6-4 Copybook for array and list sample*

```
01 request.
          02 drivers Occurs 10 Times.
              03 age pic S9(5).
              03 name pic X(20) value SPACE.
          02 vehicles-Num pic 9(9).
          02 vehicles Occurs 10 Times.
              03 vehicleId pic X(20) value SPACE.
              03 vehicleValue usage COMP-2.
```

### Mapping the basic Java type

Table 6-1 lists the Java to COBOL mapping.

*Table 6-1 Java to COBOL mapping*

| Java type | Default COBOL mapping | Configuration |
|---|---|---|
| byte | S9(3) | Sign and length<br>USAGE BINARY, PACKED-DECIMAL, COMP-5 |
| short | S9(5) | |
| int | S9(10) | |
| long | S9(18) | |
| java.math.BigInteger | S9(18) | |
| float | COMP-1 | Sign and length<br>USAGE BINARY, PACKED-DECIMAL, COMP-5, and COMP-1 |
| double | COMP-2 | |
| java.math.BigDecimal | S9(9)V9(8) | |
| java.lang.String | X(20) | X/N, length |
| java.util.Date | 9(8) [*yyyyMMdd*] | 9/X, date format |
| boolean | | |

You can change the default Java to COBOL mapping in the **COBOL Type Settings** tab of the **COBOL Management** property panels. These panels are accessed by right-clicking the project and choosing **properties.**

The following classes are unsupported Java types:

- Any Java classes, except the classes that are listed in Table 6-1
- The java.lang.Object class
- Classes that are defined in another BOM entry

## 6.2 Extending the capability of the rule execution with BOM methods

The Business Action Language (BAL) that is used to define rules is flexible and extensible. Generally, business rules are written from a vocabulary that is based on the structure of the data that is passed in, for example:

```
If the age of the borrower is less than 18
then ……
```

It is also possible to verbalize methods to be invoked from rules. These methods can come from either methods from the imported Java XOM classes or be defined directly in the BOM as virtual methods, for example:

```
public void rejectTheLoan()
{
    this.approved = false;
}
```

The method in this example can be verbalized as:

```
If the age of the borrower is less than 18
then reject the loan ;
```

Here, the method “reject the loan” can be used in place of the BAL statement:

```
make it false that the loan is approved ;
```

This approach greatly simplifies the rule authoring experience.

Although BOM methods are useful, ensure that the business decision can still be managed by the business and that the decision is still reusable across multiple platforms. The next section, 6.2.1, “Preferred practices for using virtual methods” on page 115, describes several of the preferred practices for BOM methods and shows you an example of using them.

## 6.2.1 Preferred practices for using virtual methods

To the IT-focused decision developer, the BOM methods might appear to be an attractive way to augment the capabilities of a business decision. However, you can negate the value of externalizing a business decision into a business rule engine if you do not use the BOM methods correctly. This section describes a few of the preferred practices to help you avoid misusing the BOM methods.

### Do not bury business logic in the business decision

When you realize that you can access custom Java code from within a business decision call, it can be tempting to add business logic to the decision. Adding business logic to the decision is generally a bad idea. This statement might seem counterintuitive because often the terms business logic and business rules are used interchangeably. When looking at the business decisions, you must consider them only the *rules* part of the business application.

Figure 6-1 shows a simplified representation of a business application.

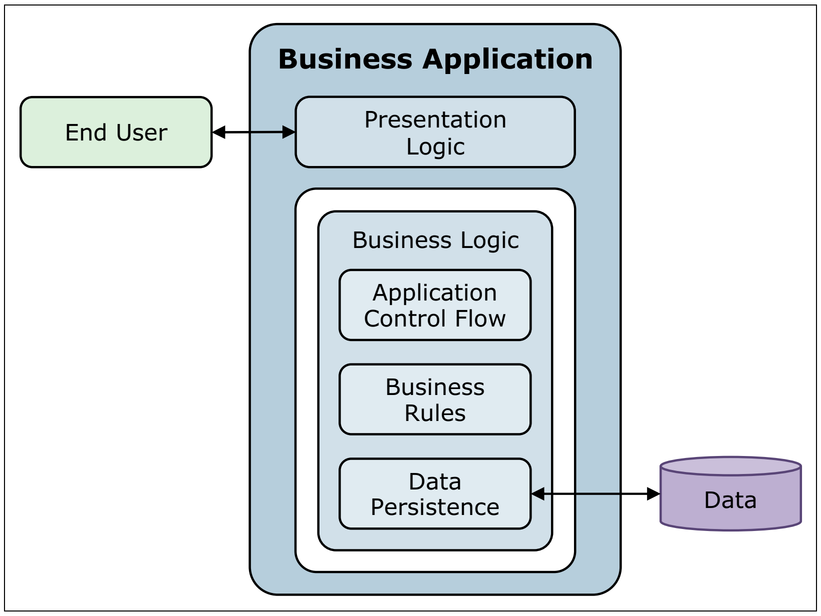


*Figure 6-1 The structure of a business application*

Figure 6-1 on page 115 shows that a business application is normally made up of the following elements:

- Presentation logic handling the user interaction
- Application control flow handling the flow of the logic through the application
- Business rules that are the implementation of the business behavior
- Data persistence layer that handles interaction with data sources

It is important to point out that the business rules do not interact with anything outside of the application. If you want to be able to hot-deploy new versions of decisions, you must be certain that changes to the business rules cannot break the application. If the business rules call out to external data sources (see 6.3, “Augmenting ruleset parameters from external data sources” on page 117), any changes must be tested within the full application, forcing a full regression test.

If the changes to the business rules change only the business behavior within the application, you can test the business rules in isolation to the full system. You only test to ensure that the rules implement the business requirement correctly. Ensure that the rules do not cause an issue for the application, for example, by forcing a divide by zero error. This level of testing is only appropriate if the business rules are encapsulated within the application.

### Do not add platform-specific logic if sharing rules

In most cases, Java is platform-independent. However, it is still possible to code Java methods that only run in certain environments. For example, the JzOS Java libraries, which are part of the base Java Runtime Environment (JRE) 6.0 for z/OS, provide a collection of methods that are useful when coding Java on the mainframe. They contain methods for accessing z/OS resources and formatting data, plus other useful features. After you code a BOM method that uses the JzOS libraries, this business decision cannot be reused on a distributed platform. The only way to share is to create either two versions of the Java XOM or two separate rulesets, each containing a separate implementation of the BOM method, depending on where the method is coded. Both of these options lead to greater complexity in the rule management that is required to keep consistent decisioning across the platforms.

### Use BOM methods sparingly

One of the key values of externalizing your business decisions as business rules is the ability to author them in natural language, making them accessible to the business team. If you use too many BOM methods, the result is recoding the business decision from the application into BOM methods in the business rules rather than as BAL rules that are accessible. In the extreme case, it is possible to code so much of the logic in BOM methods that the externalized business decision is no more accessible to the business team than the original business application from which the decision was extracted. The key is to use BOM methods sparingly where they add value to the rule authoring by simplifying the language and the logic required to author the decision.

BOM methods are useful in the following examples:

- Coding a formula that does not change but is used repeatedly within the decision.

  One example is calculating the after-tax income of a client where the tax amount is available as a variable to the BOM method.

- Simplifying the language that is required to perform a business operation to abstract from the data model.

  An example of this BOM method is where the business user has the ability to say:

  ```
  reject the loan
  ```

  Rather than having to know the relevant part of the data model to alter to create this behavior.

- Handling more complex data structures within the data model.

  The XOM model can contain more complex data structures, such as ordered lists. You can use a simple BOM method, such as addMessage(), to isolate the business user from this complex data structure.

## 6.3 Augmenting ruleset parameters from external data sources

Sometimes, decisions can only take place when data from DB2 or Virtual Storage Access Method (VSAM) is obtained to augment the ruleset parameters. In this case, there are some preferred practices that should be followed when obtaining the data.

### 6.3.1 Preferred for providing rule execution data

When designing a business rule application the preferred practice is to obtain all the required data from ruleset parameters, provided by the client application. This maintains the flexibility and portability of the rule application. It also optimizes performance, preventing the need to contact a separate data provider during execution.

Occasionally in some, less common, situations the client application cannot access the data needed to make the decision. In this case, it cannot be passed to the rule engine in ruleset parameters.

Again, it must be stressed that alternative approaches to providing data should be considered a last resort after careful consideration that the data is required in making the decision and that it cannot be provided by the client application.

### 6.3.2 Approaches to providing data from external sources

The approach that you take to access data from an external source depends on the following factors:

- The location of the data
- How frequently rule execution requires the data
- The computational cost of accessing the data via different approaches

The following sections provide information about the different ways external data can be provided to a ruleset in addition to the data provided by the client application. These approaches should only be taken after careful consideration that the data cannot be provided by the ruleset parameters.

#### Provide rule execution via a wrapper program

This solution is applicable when the required external data can be accessed from the environment on which the client (COBOL or PL/I) application is running, but you do not want to load it into that client application.

The wrapper application is called by your client application instead of calling the rule engine directly and does the following:

1. Receives Input and Output ruleset parameter data for the rule from the client application. Can optionally receive details of the RuleApp and ruleset (if the same program is used for multiple different RuleApps and rulesets).
2. Obtains the external data required, for example, from a database, data set, or file.
3. Calls the rule engine, providing the parameter data sent from the client application combined with external data obtained.
4. Passes output parameters back to the client application.

The advantages of this approach are as follows:

- The same wrapper program could be used by multiple different client applications, possibly calling multiple different RuleApps and rulesets. These applications would share a common need to provide external data to the rules that they call.
- The rule application is not modified and so rule execution performance is not affected.
- The rule application does not perform any z/OS specific function and so remains portable to a distributed environment.

Because of these advantages, this is the preferred approach to providing external data to a ruleset. This approach, however, is not always possible:

- The client environment might not allow access to the data
- The size of the data might affect performance if it was passed via a copybook. This is especially relevant if the data is only required intermittently. Passing it every time would unnecessarily affect performance.

If using a wrapper program is not possible, one of the approaches covered in the next sections could be considered, where the data is obtained during rule execution.

## Provide external data by using a XOM method

When a COBOL or PL/I client program calls a ruleset, the Operational Decision Manager marshaller code creates an instance of the Java XOM and populates it with the data the client program sent.

The XOM may have originated on the distributed platform for use with Java client applications, or it may have been automatically generated based on a COBOL copybook or PL/I include. In either case, the Java XOM code contains getter and setter functions that map to the content of the COBOL copybook or PL/I include.

To provide external data, you can add getter methods to the XOM that are not mapped to the COBOL copybook or PL/I include. Instead, these methods can perform whatever function you want in which to obtain data from an external source. No setter method should be defined.

The next time that you perform the “BOM update” function, by right-clicking the BOM, these new methods will be mapped to BOM class members and can be verbalized for use in rules.

Following are some examples of external data sources that you can access from XOM methods:

- A database, via Java Database Connectivity (JDBC)
- A file on the UNIX System Services file system
- A VSAM file, by using a JzOS feature (see the sample in Appendix A, “Calling out from a ruleset to a Virtual Storage Access Method file to augment data” on page 227)

The advantage of this approach is that when the data is only required occasionally by the rules, it need only be accessed on those occasions (by calling the XOM method). Providing the data in a conventional way via the ruleset parameters requires sending it on every rule execution.

Following are the disadvantages of this approach:

- You reduce the portability of the ruleset if it uses code that is specific to z/OS (for example, JzOS).
- If the data is accessed frequently, and could have been provided by the client application, performance and maintainability are unnecessarily impacted.

### Provide external data using a BOM virtual method

Instead of accessing external data through code in the XOM, it can be included in the BOM by using a virtual method. This is a method that is included in the BOM, but is not mapped to any XOM method.

This option might be preferred if:

- The XOM is shared between multiple rule applications but only one application has a need for the extra data.
- Responsibility for the XOM lies with another part of the organization.

Use the following steps to create a BOM virtual method:

1. Create a new empty BOM entry (by right-clicking the BOM folder in the Rule Explorer).
2. Create a class within this new BOM entry.
3. Expand the **BOM to XOM Mapping** section of the new class and enter the “Execution Name” as void.
4. In the **Members** section of the class, select **new** to define a new member.
5. In the **New Member** panel:
   a. Set the radio box **Method**.
   b. In the type field, browse for the type of the external data to be returned by this method.
   c. In the table at the bottom, use **Add** to add parameters that are required by the method; for example, information that forms the lookup for the external data.
   d. Press **Finish** to create the new member.
6. Double-click the member that you created, in the table, to open the editor view for it.
7. Check the boxes for **Static** and **Final**.
8. Expand the **BOM to XOM Mapping** section of the member editor and enter the IBM ILOG® Rule Language (IRL) code to access the external data, finishing with a return statement that passes the result back.

## 6.4 Considerations for sharing rules between z/OS and distributed applications

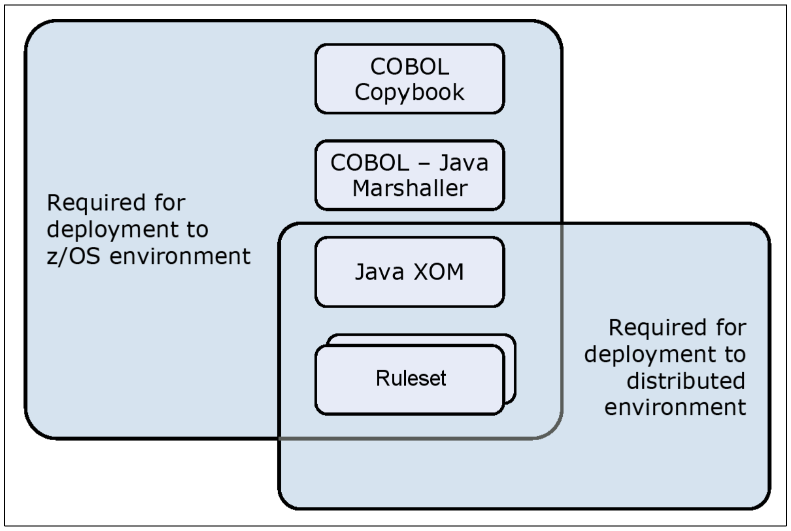
One advantage of externalizing your business decisions is that you can identify decisions that are duplicated across multiple applications. The next logical step is to remove the duplication and manage the duplicated decisions as one decision to ensure consistency across the solution. Situations can occur where a decision is deployed for both a z/OS application and a distributed application. This type of deployment is possible, although certain considerations exist.

## 6.4.1 Sharing a COBOL or PL/I-based project with Java applications

When you start from a COBOL copybook or PL/I include file as your definition for the data that is passed into the decision (the XOM), the tooling initially generates a Java representation of the COBOL data structure. These Java objects are used at run time by the decision server for evaluating the business rules. After you import the copybook and develop your rules, the following artifacts are left:

- The copybook
- A Java XOM project that represents the data from the copybook
- The Rule Project, which contains the marshaller and one or more rulesets that define the rules in the decision

All these artifacts are required to deploy the rules to the zRule Execution Server for z/OS environment, as shown in Figure 6-2.


COBOL Copybook
COBOL – Java Marshaller
Required for deployment to z/OS environment
Java XOM
Ruleset
Required for deployment to distributed environment
*Figure 6-2 Artifact deployment*

If you want to reuse this decision in a distributed environment, deploy the Java project that was created from COBOL or PL/I and the rulesets together to a distributed version of Operational Decision Manager. In this case, you use the standard Java APIs to access the decision. The client passes the data into and out of the decision using the Java XOM objects that are generated from the COBOL copybook.

You must not edit or change the generated Java classes in any way. Any change to the COBOL copybook requires the regeneration of the Java XOM. Any changes that you made are lost. In this example, consider the COBOL copybook as the master copy of the data model. Any required changes must be made to the COBOL copybook, and all other artifacts regenerated.

## 6.4.2 Sharing a Java BOM-based project with COBOL applications on z/OS

Section 6.1, “Starting from an existing Java based BOM project” on page 112, describes the process of enabling a Java BOM-based project for use with COBOL applications on z/OS. When planning to enable a Java based BOM project, it is important to consider the restrictions on the Java types that can be supported in this process. It is also important to note that new artifacts are created in this process. The most important artifact to the run time is the marshaller, which converts between the native data structures and the Java used at run time. This artifact must be deployed to the zRule Execution Server for z/OS run time with the Java XOM and ruleset projects.

When any changes are made to the Java XOM or to the generated BOM, you must rerun the process to update the COBOL artifacts to synchronize them with the Java changes. In this case, consider the Java XOM as the master data model. You must not change the COBOL copybook after it is generated.

When using an existing Java project as a starting point to deploy to z/OS, ensure that no platform-specific code is in the Java XOM. Rules can invoke methods that exist on classes in the Java XOM. Ensure that if any methods are used in the rules, the methods do not invoke any platform-specific code. Java code is independent of any platform. However, Java code can become specific to a platform if it tries to access a data source that exists only in the solution on particular servers. For example, the solution might use a client record database that is hosted locally to the decision execution on IBM AIX® server1, but is not accessible to z/OS server2 due to the firewall configuration.

## 6.5 Authoring considerations for performance

When authoring the rules, consider the implications on the performance of the decision. For more information, see the IBM Redbooks publication *Proven Practices for Enhancing Performance: A Q & A for IBM WebSphere ILOG BRMS 7.1*, REDP-4775, which provides guidance about configuring and authoring for performance.

# Part 2

# System configuration

Multiple runtime environments are possible with Operational Decision Manager for z/OS. This part describes these environments and how to configure them. It contains the following chapters:

- Chapter 7, “Prerequisites and considerations before you start” on page 125
- Chapter 8, “zRule Execution Server for IBM z/OS stand-alone server” on page 131
- Chapter 9, “zRule Execution Server for z/OS embedded server” on page 147
- Chapter 10, “Configuring IBM CICS to work with Operational Decision Manager” on page 153
- Chapter 11, “Configuring IBM IMS to work with Operational Decision Manager” on page 169
- Chapter 12, “Liberty Application Server on IBM z/OS” on page 173
- Chapter 13, “Configuring IBM WebSphere Optimized Local Adapters support” on page 183
- Chapter 14, “Configuring Decision Warehousing” on page 193
- Chapter 15, “Configuring the Rule Execution Servers for IBM z/OS console with virtual IP addressing” on page 203
- Chapter 16, “Configuring Operational Decision Manager to collect execution data using SMF” on page 213
- Chapter 17, “Problem determination” on page 223

# 7 Prerequisites and considerations before you start

This chapter provides an overview of the teams that are needed to complete a configuration of Operational Decision Manager on z/OS. This chapter also includes a checklist to be completed with the client values before embarking on a configuration.

The following topics are covered in this chapter:

- 7.1, “Runtime environments on z/OS” on page 126
- 7.2, “Teams needed for installation and configuration” on page 128
- 7.3, “Gathering the customizable information” on page 129

## 7.1 Runtime environments on z/OS

The runtime environments on z/OS all support the COBOL or PL/I execution object model (XOM), including the Rule Execution Server running on WebSphere Application Server on z/OS.

**Important:** However, the COBOL or PL/I XOM capability is not supported by the rule engine running on WebSphere Application Server on a distributed platform.

Figure 7-1 shows the runtime environments that can be configured on z/OS:

- zRule Execution Server for z/OS (zRES) hosted on CICS
- Stand-alone zRES for z/OS
- Rule Execution Server (RES) hosted on WebSphere Application Server

All of these run times support the COBOL or PL/I XOM capability.

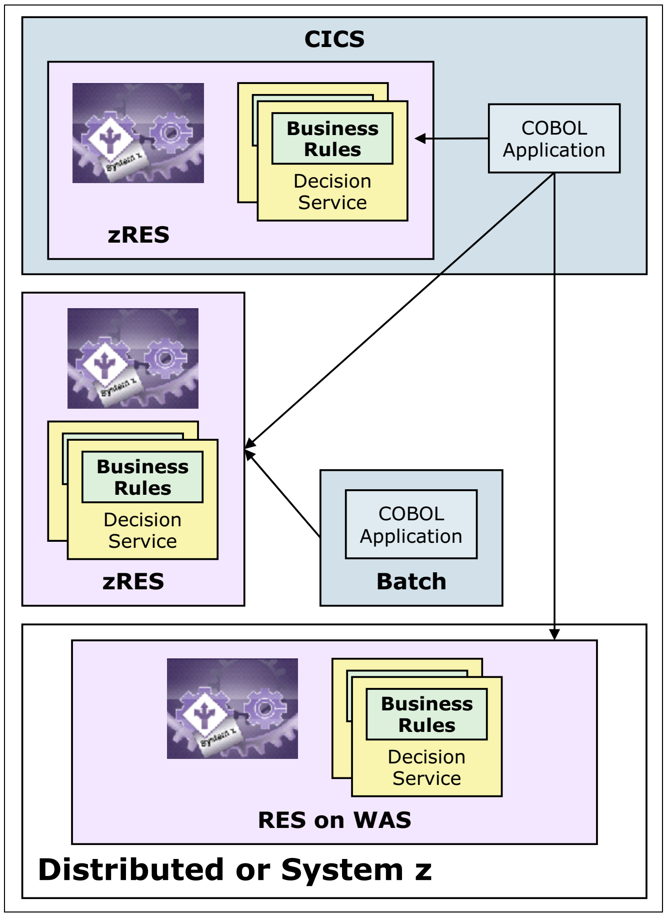


*Figure 7-1 Runtime configurations on z/OS*

### 7.1.1 Configuring the run times

You set up each runtime configuration by changing the related parameter values that are grouped into a number of partitioned data set (PDS) members as listed in Table 7-1 on page 127.

*Table 7-1 Configuration parameters*

| Member name | Description |
|---|---|
| HBRBATCH | Used to configure the batch client programs to access the zRule Execution Server. |
| HBRCICSD | DB2 parameters when using zRule Execution Server in a CICS JVM Server. |
| HBRCICSJ | Contains variables that are used when executing zRES in CICS JVM server. Also used when connecting through Distributed Program Link (DPL) when using the zRule Execution Server in a remote CICS JVM Server. HBRTARGETRES determines whether the zRule Execution Server is local or remote. |
| HBRCICSZ | Used when connecting a CICS Region to a zRule Execution Server group. |
| HBRCMMN | Common parameters for zRule Execution Server for z/OS. The language of the Rule Execution Server environment, the log level tracing, and the console communication parameters. |
| HBRCTRL | The control statements that tell the HBRUUPTI job which execution environments to create. Uncommenting these out will create those specific working data sets. |
| HBREMBED | Used when embedding the zRule Execution Server inside of a batch job's address space. |
| HBRPSIST | Member that defines the type of persistence used by the zRule Execution Server instance. This can be either DB2 or file system-based that is in UNIX System Services. |
| HBRINST | Custom configuration values for the configurations. |
| HBRSCEN | Input for the Miniloan sample. |
| HBRWOLA | The member that defines the WebSphere Optimized Local Adapters (WOLA) connection for Operational Decision Manager to connect a COBOL program to a WebSphere Application Server instance of the Rule Execution Server. |

For more information about the descriptions of the parameters in these members, see the *z/OS configuration and runtime variables* topic in the IBM Operational Decision Manager Version 8.7 IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/con_ds_jcl_and_runtime_vars.html

Each configuration parameter is updated with the site-specific values when the HBRUUPTI job is used to configure a zRule Execution Server for z/OS instance. Use the HBRUUPTI job to help configure an Operational Decision Manager z/OS instance. This job takes the client values from the member HBRINST and stamps them into the related configuration parameter members and configuration JCL jobs for that runtime instance. The configurations that are created are based on the HBRCTRL member.

See Appendix B, "Configuring runtime values by using variables defined in HBRINST" on page 231 for the client values that need to be gathered before you configure Operational Decision Manager for z/OS.

## 7.1.2 Prerequisite checklist

Use Table 7-2 on page 128 to check that the z/OS system is at the correct prerequisite level. For the current information, see the IBM Decision Center for z/OS 8.7 on z/OS website:

http://www.ibm.com/software/reports/compatibility/clarity-reports/report/html/softwareReqsForProduct?deliverableId=1399048306928&osPlatforms=z/OS

*Table 7-2 Prerequisites*

| Item | Value |
|---|---|
| z/OS level | 1.13 or higher |
| Java level + service | 6.0.1, 7, 7.1 all 64-bit support |
| DB2 + service | 10 or 11 |
| CICS + service | ► CICS 3.2<br>► CICS 4.2<br>► CICS 5.1<br>► CICS 5.2 |
| WebSphere Application Server level | ► V8 with Fix Pack 4<br>► V8.5 |
| Operational Decision Manager | ► HDM8710 - ODM Base z/OS (mandatory)<br>► HDM8711 - ODM and BR Common (mandatory)<br>► JDM8712 - Events Component z/OS<br>► HDM8713 - Rules Component z/OS<br>► JDM8714 - Decision Center z/OS |

## 7.2 Teams needed for installation and configuration

Before installing and configuring Operational Decision Manager on z/OS or Operational Decision Manager on WebSphere Application Server for z/OS, it is necessary to involve various groups that administer the products involved. Without these people, the configuration is likely to be unsuccessful. In particular, the jobs for creating the database can be precustomized by following the initial customization steps from the product documentation, but the customized jobs might not be applicable to every site. These JCL jobs are a suggested way of working and are examples that might require editing further for each location.

When considering an Operational Decision Manager environment, a number of infrastructure teams might need to be consulted. The following team functions can be grouped into one or many teams:

- Installation group: Responsible for installing products by using SMP/E for z/OS
- System programmers: Responsible for creating started tasks, subsystems, z/OS file systems, datasets, and system configuration
- Security managers: Responsible for setting up IBM RACF® or equivalent classes and profiles
- Database administrators: Responsible for creating the DB2 repositories
- WebSphere Application Server administrators, if running on WebSphere Application Server
- CICS administrators, if running the rules engine with or in CICS
- IMS administrators, when running with IMS

## 7.3 Gathering the customizable information

After help is committed from the groups that are mentioned in 7.2, “Teams needed for installation and configuration” on page 128, gather all the information that is required to customize the JCL jobs. The task of customizing the JCL jobs is done by the HBRUUPTI job and uses the input that is supplied by the HBRCTRL and HBRINST members.

See Appendix B, “Configuring runtime values by using variables defined in HBRINST” on page 231 for details of the variables that are available in HBRINST.

8

# zRule Execution Server for IBM z/OS stand-alone server

This chapter describes the process of setting up a stand-alone zRule Execution Server for z/OS (zRES) server. This chapter also describes how the different types of database connections are set up from the zRES server.

The following topics are covered in this chapter:

## 8.1 Running on z/OS stand-alone

This section describes how to run the rule engine on z/OS to consume batch work. The following components make up the zRES:

- A stand-alone address space that hosts the rules engine
- A zRES console to administer the zRES server
- A database to hold the rules

Figure 8-1 shows the runtime environment of a stand-alone zRule Execution Server for z/OS.

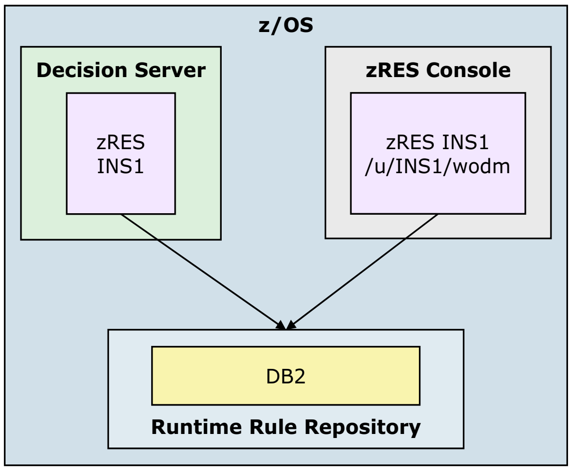


*Figure 8-1 zRES stand-alone server*

### 8.1.1 Configuring the stand-alone zRule Execution Server for z/OS

To configure the stand-alone zRule Execution Server for z/OS, you must edit the values in the HBRINST and HBRCTRL members and run the HBRUUPTI job. This action creates a set of data sets that are configured for this zRES instance.

For example, the output from HBRUUPTI can produce the runtime configuration data sets that are shown in Table 8-1 and Example 8-1 for one zRES instance.

*Table 8-1 zRES instance configuration data sets*

| Data set | Description |
|---|---|
| ++HBRWORKDS++SHBRJCL | Instance configuration jobs |
| ++HBRWORKDS++SHBRPARM | Instance configuration values |
| ++HBRWORKDS++SHBRPROC | Configured zRule Execution Server for z/OS started tasks |

*Example 8-1 Example zRES runtime configuration data sets*

```
++HBRWORKDS++.HBR1.SHBRJCL
++HBRWORKDS++.HBR1.SHBRPARM
++HBRWORKDS++.HBR1.SHBRPROC
```

This step is repeated for each zRule Execution Server for z/OS instance that you want to create. Each zRule Execution Server for z/OS has a unique identifier that is given by the HBRSSID value, which was determined in the HBRCTRL member.

### Defining the zRule Execution Server for z/OS instance working directory

In addition to a set of data sets for each zRule Execution Server for z/OS instance, each instance also has a file system location. The run time uses the file system location, which is called a *working directory*.

Running the HBRCRTI job from the instance data set SHBRJCL sets up the working directory in z/OS System Services for the zRule Execution Server for z/OS instance.

## 8.1.2 Creating data sets for the zRule Execution Server for z/OS instance

This section describes how to create the data sets that are changed for the zRule Execution Server for z/OS instance.

### Customizing the HBRINST member of SHBRPARM

This section details which values must be updated. For more information, see Appendix B, "Configuring runtime values by using variables defined in HBRINST" on page 231. This appendix contains tables that explain how to customize the values in SHBRPARM(HBRINST) for your system environment.

**Preferred practice:** A preferred practice is to copy the target library SHBRPARM partitioned data set (PDS) members HBRINST and HBRCTRL as new members within the PDS, for example, HBRINSTD and HBRCTRLD. The HBRUUPTI job is modified to point to these instances of HBRINSTD and HBRCTRLD.

### Setting up database persistence

When setting up the first zRES server, several values within the HBRINST member need to be updated. Example 8-2 lists the variables that are updated in the first instance using database persistence. The jobs that need to be ran to create the DB2 persistence layer are in the following bulleted list. Have your DB2 Administrator inspect and change the following jobs as required:

- HBRDSCDB
- HBRDSXOM
- HBRDSCTR
- HBRDSCDR
- HBRDSGRN

**Editing the DB2 jobs:** Pre-version 8.7.1, if editing these tables, turn CAPS OFF because there is a case-sensitive parameter in HBRDSCDB for the console to connect to the database.

For more information about each of these variables, see Appendix B, "Configuring runtime values by using variables defined in HBRINST" on page 231.

**Type 2 connection:** By default, zRES establishes a type 4 connection. For details about how to use a type 2 connection, see 8.3, "Setting up the database connection as a Type 2 connection" on page 145.

*Example 8-2 List of variables that are used for database persistence*

```
DB2HLQ
DB2RUNLIB
DB2SUBSYSTEM
```

*DB2LOCATION*
*DB2VCAT*
*DB2ADMIN*
*DB2SCHEMA*
*RESDATABASE*
*RTSDATABASE*
*EVDATABASE*
*RESSTOGROUP*
*RTSSTOGROUP*
*EVSTOGROUP*
*DB2TABLEBP*
*DB2INDEXBP*
*DB2LOBBP*
*DB2SAMPLEPROGRAM*
*DB2SAMPLEPROGRAMPLAN*
*DB2BP4K*
*DB2BP8K*
*DB2BP32K*
*DB2USER*
*DB2PSWD*
*DB2GROUP*
*DB2SERVNAME*
*DB2PORT*
*DB2JARLOCN*
*DB2NATIVELOC*

## 8.1.3 Creating the working data sets using HBRUUPTI

The HBRUUPTI member that is within the ++HBRHLQ++.SHBRJCL data set uses the values in the HBRINST member to populate the SHBRJCL, SHBRPARM, SHBRPROC, and SHBRWASC data sets that are changed to your system environment.

### Changing HBRUUPTI

You must perform the following steps to change HBRUUPTI to create the new working data sets for the zRule Execution Server for z/OS server. Customize the HBRINST data set to your system environment by using the tables in Appendix B, "Configuring runtime values by using variables defined in HBRINST" on page 231.

The preferred practice is to copy the target library SHBRPARM member HBRINST as a new member within the SHBRPARM PDS and to create a new PDS with a name similar to HBRINST (for example, HBRINSTD as outlined from "Customizing the HBRINST member of SHBRPARM" on page 133.

To change HBRUUPTI, follow these steps:

1. Update the following line in HBRUUPTI that shows the target library high-level qualifier (HLQ) that is set to the value HBRHLQ. Update this line with the value of your HBRHLQ from Table B-1 on page 232. In this example, it is set to ODM.V8R7M1.TLIB:

   ```
   SET HBRHLQ=ODM.V8R7M1.TLIB
   ```

2. Update the INLINES line, as shown in Figure 8-2, to match where the customization member is created, which, by default, is in HBRHLQ.SHBRPARM(HBRINST). This points to the instance of HBRINST that you use for the customization. If you use the preferred practice, ensure that you update this to the correct value you have set.

```
//         SET HBRHLQ=WODM.V8R7M1.TLIB
//HBRUUPTI EXEC PGM=IKJEFT01,REGION=2M,DYNAMNBR=99
//SYSPROC DD DISP=SHR,DSN=&HBRHLQ..SHBREXEC
//INCNTRL DD DISP=SHR,DSN=&HBRHLQ..SHBRPARM(HBRCTRLD)
//INLINES DD DISP=SHR,DSN=&HBRHLQ..SHBRPARM(HBRINSTD)
//SYSTSIN DD *
```

*Figure 8-2 Changing the INLINES line*

3. Submit the job to create the working data sets for the zRule Execution Server instance. This job creates the following data sets:

   – ++HBRWORKDS++.SHBRJCL
   – ++HBRWORKDS++.SHBRPARM
   – ++HBRWORKDS++.SHBRPROC

   For this example, if the HBRSSIDLIST were HBR1 and HBR2, and the HBRWORKDS was ODM.V8R7M1, the following data sets are created:

   – ODM.V8R7M1.HBR1.SHBRJCL
   – ODM.V8R7M1.HBR1.SHBRPARM
   – ODM.V8R7M1.HBR1.SHBRPROC
   – ODM.V8R7M1.HBR2.SHBRJCL
   – ODM.V8R7M1.HBR2.SHBRPARM
   – ODM.V8R7M1.HBR2.SHBRPROC

## 8.1.4 Creating the working directories in UNIX System Services

After submitting the HBRUUPTI job, navigate to the following PDS, and open the job HBRCRTI:

```
++HBRWORKDS++.SHBRJCL
```

This job runs a script, **`hbrcrtin.sh`**, that is in the ++HBRINSTPATH++, which is set in Table B-1 on page 232. This job creates the ++HBRWORKPATH++ directory in the UNIX System Services, which contains the following directories:

- `config`
- `logs`
- `res_data`
- `res_xom`
- `work`

The `config` directory contains the XML files that are required for the zRule Execution Server to start, including the run time and the console.

**Preferred practice:** By default, this job mounts all of the files to your system's root file system. To avoid this, create a file system before running this job at the ++HBRWORKPATH++ to contain the zRES configuration files. Another mount point should be created at ++HBRWORKPATH++/logs to allow the logs to not be tied to the configuration files. This way, if you are troubleshooting a job, the logs can grow in size independent of the configuration files and if it filled up the directory, it would not affect the runtime directory structures.

## 8.2 Configuring the stand-alone zRule Execution Server for z/OS

This section covers configuring the stand-alone zRule Execution Server for z/OS or one zRule Execution Server group with one console.

For every type of setup, initially configure at minimum a zRule Execution Server console as this is required for the zRule Execution Server stand-alone execution unit, the zRule Execution Server in the CICS JVM, and the embedded zRule Execution Server.

### 8.2.1 Defining a new subsystem for zRule Execution Server for z/OS

The first step for the configuration of the stand-alone zRule Execution Server for z/OS is to define the subsystem in which the new instances run. The systems programmer must perform this task.

The following **SETSSI** command must be run, where *++HBRSSID++* is the subsystem ID that was set in Table B-1 on page 232 under ++HBRSSIDLIST++. Run this command for each instance:

```
SETSSI ADD,SUBNAME=++HBRSSID++
```

For example, if HBR1 and HBR2 are in the list, this command is run for each instance. Using the example of HBR1 and HBR2 as the SSIDs, the following commands are correct:

```
SETSSI ADD,SUBNAME=HBR1
SETSSI ADD,SUBNAME=HBR2
```

### 8.2.2 Creating the started tasks (HBRXMSTR)

The next task in the configuration is to copy HBRXMSTR to the system PROCLIB for each created subsystem. The HBRXMSTR members are for the started tasks. These started tasks are attached to three parmlib members that are used for the definitions of various parameters for the zRule Execution Server. If you use the HBRUUPTI job, you do not need to modify these attached parmlib members unless your environment requires the change.

#### Adding HBRXMSTR to SYS1.PROCLIB

The next task is to copy HBRXMSTR to `SYS1.PROCLIB` (or a similar PROCLIB on your environment). When copying over the data set members, you must change the names from HBRXMSTR to ++HBRSSID++MSTR. Follow these steps:

1. Copy the ++HBRWORKDS++.SHBRPROC(HBRXMSTR) to `SYS1.PROCLIB`(++SSID++MSTR).
2. Repeat the process for HBRXMSTR for all members created from HBRCTRL.

**SYS1.PROC:** `SYS1.PROCLIB` is the default. You need to change it to match your environment. ++HBRSSID++ was set in Table B-1 on page 232.

### Authorizing the server instance as a started task

Authorize all ++HBRSSID++MSTR as started task procedures executing with the same user ID. Use the following commands:

```
RDEFINE STARTED ++HBRSSID++MSTR.* STDATA(USER(<HBRSSID_USER>)
GROUP(<HBRSSID_GROUP>)
```

*HBRSSID_USER* is the server user ID and *HBRSSID_GROUP* is the RACF security group name that is provided to you by your security administrator.

### The started task definitions

Whether you are starting the stand-alone zRule Execution Server for z/OS started task or the CICS zRule Execution Server for z/OS started task, both started tasks require that the configuration parameters are provided to the job. The parameters are provided by the DD card HBRENVPR on each started task. The DD card HBRENVPR specifies the input parameter members.

## 8.2.3 Securing the zRule Execution Server for z/OS for z/OS resources

With Operational Decision Manager, you can secure the resources, files, and functions with RACF. This section describes how to create this security for the server using RACF.

### Security options

If running the zRule Execution Server for z/OS in production, you might want to secure all or part of the zRule Execution Server for z/OS resources. However, if you plan to run the server in a testing environment, you might want security disabled. You can use the following options for security.

Within the file system, you can secure the following resources:

- The working directory so that only the authorized user IDs can access runtime data
- The installation directory so that only the authorized user IDs can access the files that are needed to run the server

Using RACF, you can secure the following resources:

- You can secure the server resources that you use to perform the following tasks:
  - Issue zRule Execution Server for z/OS commands from the z/OS console (or equivalent).
  - Sign on to the Rule Execution Server console.
  - Connect to the zRule Execution Server for z/OS to execute rulesets.
- You can secure a subset of server resources. For example, you can secure access to the Rule Execution Server console only.

### Securing access to the working directory and installation directory

The working directory contains data that includes logs from the zRule Execution Server for z/OS and configuration files. The installation directory contains the binary files that are required to run the zRule Execution Server for z/OS server. The server user ID needs to read and execute access for ++HBRWORKPATH++ and ++HBRINSTPATH++ which are the zRule

Execution Server for z/OS work path, and the Operational Decision Manager installation directory.

### Creating the RACF classes for securing server resources

You can manage zRule Execution Server for z/OS by using RACF classes. You must create the three RACF classes by using the ++HBRWORKDS++.SHBRJCL(HBRCRECL) job. To secure the resources for the zRule Execution Server for z/OS instance, your RACF administrator must run the HBRCRECL job. This job can be run from any of the members within the ++HBRSSIDLIST++.

### Creating the RACF classes

Using RACF, you can secure the following information:

- Ask the RACF administrator to run the HBRCRECL job or extract the code to use the preferred execution methods to perform the following tasks:
  - Issue zRule Execution Server for z/OS commands from the z/OS console (or equivalent).
  - Sign on to the Rule Execution Server console.
  - Connect to the zRule Execution Server for z/OS to execute rulesets.
- You can secure a subset of server resources. For example, you can secure access to the Rule Execution Server console only.

When your RACF administrator runs the HBRCRECL job, the job creates three RACF classes: HBRADMIN, HBRCONN, and HBRCMD. Table 8-2 explains the characteristics of these classes.

*Table 8-2 RACF classes created by ++HBRWORKDS++.SHBRJCL(HBRCRECL)*

| Class | Description |
|---|---|
| HBRADMIN | This class controls whether server security and security for specific server resources are enabled or disabled. |
| HBRCONN | This class specifies the user IDs that are authorized to connect to the zRule Execution Server for z/OS and to execute rulesets. This class is ignored if server security is disabled. |
| HBRCMD | This class specifies the user IDs that are authorized to issue zRule Execution Server for z/OS commands, such as `SET, PAUSE`, or `RESUME` from the z/OS console (or equivalent). This class is ignored if server security is disabled. |

**POSIT:** The supplied JCL in HBRCRECL gives a POSIT value of 128. Change POSIT, as required, to match your security environment requirements.

After running the HBRCRECL job, give the server user ID read access to the class profile by using the following commands:

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(<HBRSSID_USER>) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

In this example, <*HBRSSID_USER*> represents the server user ID, which is the ID under which the server runs.

## Disabling types of security

In Operational Decision Manager, you can optionally disable all types of security or parts of the security. When the HBRADMIN class exists, security is enabled on all zRule Execution Server for z/OS instances. Security can be enabled and disabled, as required. On a test system, you might want no security on the instance so that you can test more freely, but you do not have to disable the security to all instances that are used.

To disable levels of security, you must apply separate profiles to the HBRADMIN class. Table 8-3 lists the profiles that can be added to the HBRADMIN class by using the following commands:

```
RDEFINE HBRADMIN <RESOURCE_PROFILE> UACC(NONE)
SETROPTS RACLIST(HBRADMIN) REFRESH
```

*Table 8-3 Resource profiles to disable parts of security on the zRule Execution Server*

| Resource profile | Description |
|---|---|
| ++HBRSSID++.NO.SUBSYS.SECURITY | This profile disables all security for a particular server instance. If server security is disabled, HBRCONN and HBRCMD classes are not used. |
| ++HBRSSID++.NO.CONNECT.SECURITY | This profile disables connection security for a particular server instance, but it maintains other types of security. |
| ++HBRSSID++.NO.COMMAND.SECURITY | This profile disables command security for a particular server instance, but it maintains other types of security. If you disable command security, any user can issue a zRule Execution Server for z/OS command from the z/OS console. |

Replace ++HBRSSID++ with a value from the *++HBRSSIDLIST++* variable. Repeat for each server that is listed in the *++HBRSSIDLIST++* variable for which you want to disable security. Table B-1 on page 232 has details about the *++HBRSSIDLIST++* variable.

For more information, see the *Managing server security* topic in the IBM Operational Decision Manager Version 8.7 IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/tpc_ds_manage_zres_security.html

Continue to one of the following sections, depending on the profile that you plan to use:

- If the CONNECT profile is used, go to "Managing connection security" on page 139.
- If the COMMAND profile is used, to go "Managing command security" on page 141.

## Managing connection security

You set up connection security to ensure that only authorized user IDs can connect to the zRule Execution Server for z/OS instance to execute rulesets. Connection security uses the HBRCONN RACF class to authorize user IDs to connect to the server instance.

If the profile ++HBRSSID++.NO.SUBSYS.SECURITY or ++HBRSSID++.NO.CONNECT.SECURITY is used, the HBRCONN class is ignored.

To implement connection security, you must authorize the user ID under which the server runs and the user IDs of any applications that execute rulesets. The following steps are required for authorizing user IDs to the HBRCONN class:

1. The resource profile needs the server instance defined to the HBRCONN class. Execute the following command first to create the resource profile:

   ```
   RDEFINE HBRCONN ++HBRSSID++ UACC(NONE)
   ```

2. Give the server user ID UPDATE access to the ++HBRSSID++ resource profile by using the following command:

   ```
   PERMIT ++HBRSSID++ CLASS(HBRCONN) ID(<HBRSSID_USER>) ACCESS(UPDATE)
   ```

   **UPDATE access:** The server instance fails to initialize if the HBRCONN class does not have UPDATE access. This requirement does not affect a server instance with ++HBRSSID++.NO.SUBSYS.SECURITY, or ++HBRSSID++.NO.CONNECT.SECURITY.

3. Refresh the ++HBRSSID++ resource profile by using the following command:

   ```
   SETROPTS RACLIST CLASS(HBRCONN) REFRESH
   ```

Next, authorize the applications by running the following steps:

1. Give READ access to the ++HBRSSID++ resource profile to each user that you want to authorize. Use the following command:

   ```
   PERMIT ++HBRSSID++ CLASS(HBRCONN) ID(<USER_ID>) ACCESS(READ)
   ```

   **User IDs:** For batch jobs, <*USER_ID*> is the RACF user ID that is used by the batch job. For CICS transactions, <*USER_ID*> is the user ID that is assigned to the CICS address space.

2. Refresh the ++HBRSSID++ resource profile by using the following command:

   ```
   SETROPTS RACLIST(HBRCONN) REFRESH
   ```

### Managing console security

You use console security to ensure that there is control on the users that can access the zRule Execution Server for z/OS console. The zRule Execution Server for z/OS console security controls the ability to sign on to the zRule Execution Server for z/OS console. If security is enabled, users must enter a user ID and password to sign on.

If the profile ++HBRSSID++.NO.SUBSYS.SECURITY is used, the HBRADMIN class is ignored.

A standard set of roles exists within the zRule Execution Server for z/OS that gives access rights to users. Enable console security by assigning user IDs to roles and then authorizing the roles to access the console.

Table 8-4 on page 141 shows the profiles and the roles that they represent. The roles are listed in order of increasing authority. RESMON is the lowest authority, and RESADMIN is the highest authority. ++HBRSSID++ is the ID of the subsystem where the server runs.

*Table 8-4 zRule Execution Server for z/OS console security profiles*

| Resource profile | Role description |
|---|---|
| ++HBRSSID++.ROLE.RESMON | Users with monitoring rights are only allowed to view and explore RuleApps, rulesets, decision services, execution units (XUs), and statistics. These users are not allowed to modify these entities. They can also select a trace configuration and view and filter trace information in Decision Warehouse. This authority applies only to Rule Execution Server on WebSphere Application Server for z/OS. |
| ++HBRSSID++.ROLE.RESDEP | In addition to monitoring rights, users with deploying rights are allowed to deploy RuleApp archives to edit and remove entities (RuleApps, rulesets, decision services, Java execution object module (XOM) resources, and libraries), and to run diagnostics. |
| ++HBRSSID++.ROLE.RESADMIN | Users with administrator rights have full control over the deployed resources and access to information about the server. They can perform the following actions:<br>► Deploy, browse, and modify RuleApps, Java XOM resources, and libraries.<br>► Monitor the decision history, purge the history, and back up the history.<br>► Select a trace configuration, view and filter trace information, and clear trace information in Decision Warehouse.<br>► Run diagnostics and view server information. |

Perform the following steps to enable console security:

1. Define each resource profile, as shown in Table 8-4 to the HBRADMIN class. Use the following commands to define all three roles:

```
RDEFINE HBRADMIN ++HBRSSID++.ROLE.RESMON UACC(NONE)
RDEFINE HBRADMIN ++HBRSSID++.ROLE.RESDEP UACC(NONE)
RDEFINE HBRADMIN ++HBRSSID++.ROLE.RESADMIN UACC(NONE)
```

2. Assign each user ID to one of the resource profiles by using the following commands for the three roles:

```
PERMIT ++HBRSSID++.ROLE.RESMON UACC(NONE)
PERMIT ++HBRSSID++.ROLE.RESDEP UACC(NONE)
PERMIT ++HBRSSID++.ROLE.RESADMIN UACC(NONE)
```

3. Refresh the HBRADMIN class by using the following command:

```
SETROPTS RACLIST(HBRADMIN) REFRESH
```

## Managing command security

You use command security to ensure that only authorized users can issue zRule Execution Server for z/OS commands on the zRule Execution Server for z/OS console. Command security uses the HBRCMD RACF class to authorize user IDs to issue zRule Execution Server for z/OS commands.

If the profile ++HBRSSID++.NO.SUBSYS.SECURITY or the profile ++HBRSSID++.NO.COMMAND.SECURITY is used, the HBRCMD class is ignored.

When enabling command security on the zRule Execution Server for z/OS console, you must define a resource profile to the HBRCMD class for each command that you want to secure. Use the commands that are listed in Table 8-5 on page 142 to secure the zRule Execution Server for z/OS console commands. ++HBRSSID++ is the ID of the subsystem where the server runs.

Table 8-5 zRule Execution Server for z/OS command security profiles

| Resource profile | Command | Command description |
|---|---|---|
| ++HBRSSID++.SET.TRACE | SET TRACE | Ability to change the trace level of the zRES. |
| ++HBRSSID++.PAUSE | PAUSE | Pause the zRES from accepting new work. |
| ++HBRSSID++.RESUME | RESUME | Resume a paused zRES. |
| ++HBRSSID++.DUMP | DUMP | Request a dump from the zRES. |

To authorize users to issue zRule Execution Server for z/OS commands, perform the following steps:

1. If you want to limit any of the commands in Table 8-5 to authorized user IDs, you must define the resource profiles to the following HBRCMD class commands:

   ```
   RDEFINE HBRCMD <RESOURCE_PROFILE> UACC(NONE)
   ```

2. If you want to limit any of the commands in Table 8-5 to authorized user IDs, you must permit the resource profiles to the following HBRCMD class commands:

   ```
   PERMIT <RESOURCE_PROFILE> CLASS(HBRCMD) ID(<USER_ID>) ACCESS(UPDATE)
   ```

   **User IDs:** For batch jobs, <*USER_ID*> is the RACF User ID that is used by the batch job. For CICS transactions, <*USER_ID*> is the user ID that is assigned to the CICS address space.

3. Refresh the HBRCMD class by using the following command:

   ```
   SETROPTS RACLIST(HBRCMD) REFRESH
   ```

## 8.2.4 Starting the new instance

After completing the security setup and configurations, start the new server instance through the z/OS console.

### Authorizing the load library

If you are setting up your first instance on the logical partition (LPAR), you must authorize the load library. You must perform the following steps:

1. Add the ++HBRHLQ++.SHBRAUTH load library to the authorized program facility (APF)-authorized libraries by using the following command:

   ```
   SETPROG APF,ADD,DSNAME=++HBRHLQ++.SHBRAUTH,SMS
   ```

2. Enable program control authorization for the ++HBRHLQ++.SHBRAUTH load library by using the following two commands:

   ```
   RALTER PROGRAM * ADDMEM('++HBRHLQ++.SHBRAUTH'//NOPADCHK)
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

   Note that ++HBRHLQ++ is the product installation target library HLQ for the SHBRAUTH PDS.

### Starting a server instance

To start a new server instance, issue the following command:

```
START ++HBRSSID++MSTR
```

Replace ++HBRSSID++ with each member created by HBRCTRL to start all of the MASTER address spaces.

If the server does not start, look at the output of the HBRMSTR job to see why it did not start. The Job Step code can be searched in the IBM Knowledge Center to discover the reason why the server did not start.

Typically, the server does not start for the following reasons:

- The load library was not APF-authorized.
- ++HBRINSTPATH++ does not point to the correct UNIX System Services directory.
- If you use symbolic links on ++HBRINSTPATH++ or ++HBRWORKPATH++, these links might not link correctly. Therefore, you must verify the link.
- You did not execute the RACF security commands. Verify whether the RACF security commands were run by using the resource profile setup. Ensure that the commands executed and ensure that the user that started the server is authorized to start the server.
- The ports that were used for ++HBRCONSOLEPORT++ and ++HBRCONSOLECOMPORT++ were already in use by another application.
- The same port is used for ++HBRCONSOLEPORT++ and ++HBRCONSOLECOMPORT++.

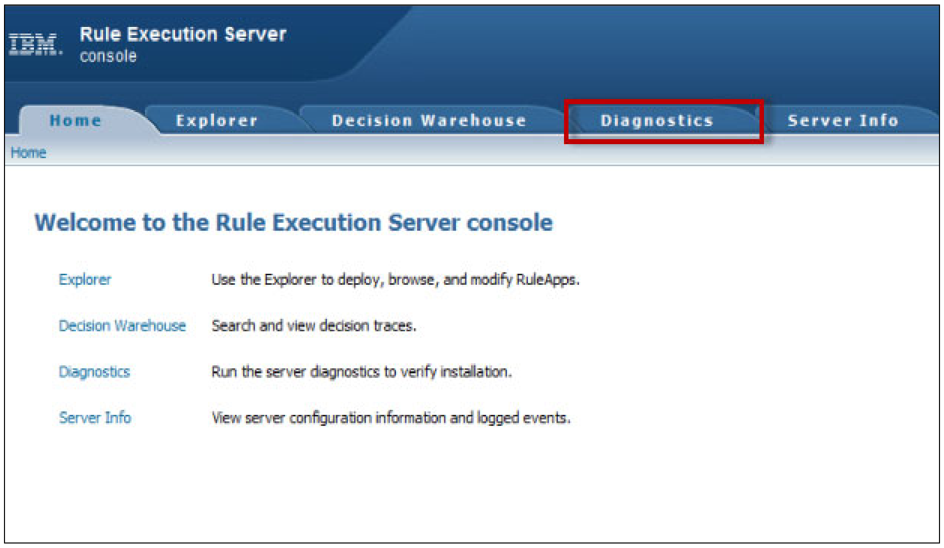
## 8.2.5 Logging on

Now the stand-alone zRule Execution Server for z/OS console is up and running zRule Execution Server for z/OS console and performing diagnostics.

Perform the following steps:

1. Using your browser, go to `http://++HBRCONSOLECOMHOST++:++HBRCONSOLEPORT++/res`. An example of this could be `http://zserveros.demos.ibm.com:34114/res.`

Figure 8-3 on page 144 shows the console after logging in from an administrator's perspective. This gives the most options from a zRule Execution Server console's perspective. You can see the rule applications and its associated resources deployed under the "Explorer" tab. From the "Decision Warehouse" tab, when turned on and executed against, you can look at transaction information captured. The "Diagnostics" tab lets you perform diagnostics from the console to verify connectivity and execution by the execution server. The "Server Info" tab displays which execution units are connected to the zRule Execution Server console. The console uses TCP/IP to manage the server connections.



Welcome to the Rule Execution Server console

| | |
|---|---|
| Explorer | Use the Explorer to deploy, browse, and modify RuleApps. |
| Decision Warehouse | Search and view decision traces. |
| Diagnostics | Run the server diagnostics to verify installation. |
| Server Info | View server configuration information and logged events. |

*Figure 8-3 The Rule Execution Server console has several tabs available for administrators. By using the Diagnostics tab, you can test that the console and an execution unit are working properly.*

Figure 8-4 shows how to execute the diagnostics test.

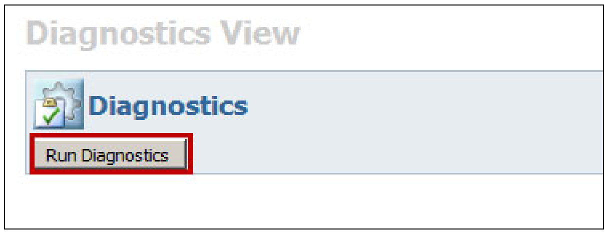
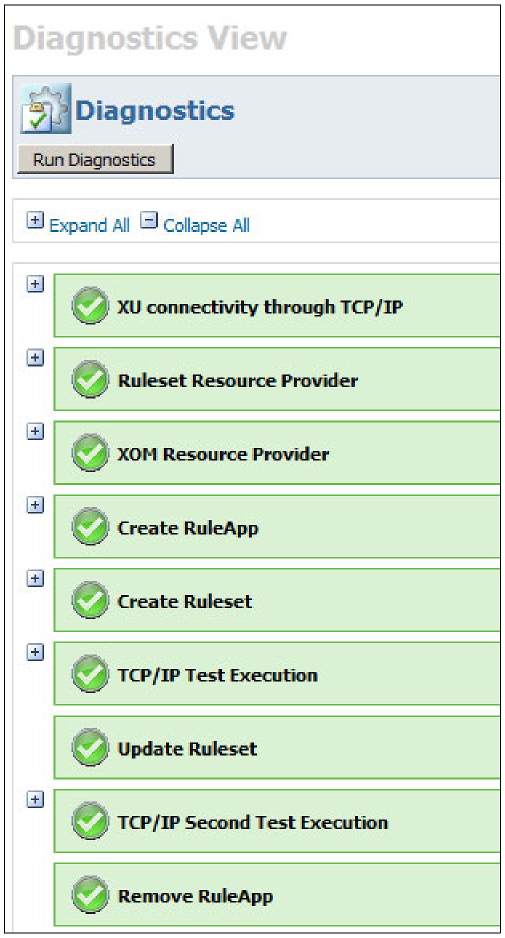


*Figure 8-4 For the diagnostics to run, click "Run Diagnostics"*

Figure 8-5 on page 145 shows a successful execution of the diagnostics test.



XU connectivity through TCP/IP

Ruleset Resource Provider

XOM Resource Provider

Create RuleApp

Create Ruleset

TCP/IP Test Execution

Update Ruleset

TCP/IP Second Test Execution

Remove RuleApp

*Figure 8-5 A successful diagnostics test.*

## 8.3 Setting up the database connection as a Type 2 connection

Connection to the database can be either a type 4 connection or a type 2 connection. See the DB2 documentation to determine which type is preferable for your environment. For more information, see the *How JDBC applications connect to a data source* topic in the DB2 10 for z/OS IBM Knowledge Center:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.java%2Fsrc%2Ftpc%2Fimjcc_cjvjdcon.htm

The zRule Execution Server in RULE, CONSOLE, or TEST mode makes connections to the database, so all need to be configured to connect to DB2.

The default database connection that is created by zRES is type 4, which is documented in the *Step 7: Configuring a DB2 persistence layer* topic of the IBM Operational Decision Manager Version 8.7 IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/tpc_ds_create_db2_persist.html

This section explains how to set up a type 2 database connection for the console and for zRES.

## 8.3.1 Setting up a type 2 configuration for the console

You need to include the DB2 library in the `STEPLIB` in the zRES PROC member. Obtain these in ++HBRWORKDS++.SHBRPROC (from where they are copied to the SYS1.PROCLIB data set, as described in 8.2.2, “Creating the started tasks (HBRXMSTR)” on page 136).

For HBRXCNSL, and for each HBRXMSTR that accesses the database, add the DB2 libraries to the `STEPLIB`, for example:

```
//STEPLIB  DD DISP=SHR,DSN=&HBRHLQ..SHBRAUTH
//         DD DISP=SHR,DSN=SYS2.DB2.V10.SDSNLOAD
//         DD DISP=SHR,DSN=SYS2.DB2.V10.SDSNLOD2
//         DD DISP=SHR,DSN=SYS2.DB2.V10.SDSNEXIT
```

## 8.3.2 Updating the database parameters in HBRPSIST

The URL that specifies the location of the database is in ++HBRWORKDS++.SHBRPARM(HBRPSIST). This needs to be updated, for example:

```
* URL associated with the database.
HBRDBURL=jdbc:db2:DSN10GP:currentSchema=ZRES;
```

## 8.3.3 Setting up the DB2 identifying file

You need to create an identifying file that indicates the SSID of the DB2 subsystem. This file can then be read by the relevant Java virtual machines (JVMs). This file needs to be an EBCDIC file in UNIX System Services, and it contains the SSID of the database. For example, for the DB2 SSID DHGP, this file contains the following line:

```
db2.jcc.ssid=DHGP
```

## 8.3.4 Updating the PARM members

You need to update the PARM members that are associated with each zRES that accesses the database. The member is HBRMSTR, and it is in ++HBRWORKDS++.SHBRPARM. For this member, perform the following steps:

1. Include a LIBRARY_SUFFIX in the PARM member that points to the DB2 libraries on System Services. Add a line that indicates the LIBPATH_SUFFIX, for example:

   ```
   LIBPATH_SUFFIX=/usr/lpp/db210/lib
   ```

2. Indicate the DB2 subsystem to the JVM by pointing the JVM at the file that is created in 8.3.3, “Setting up the DB2 identifying file” on page 146, using the **`JAVA_OPTIONS`** parameter. For example, if you use the identifying file `DB2_SSID` in the `/u/db2Id` directory, your **`JAVA_OPTIONS`** might read this way:

   ```
   JAVA_OPTIONS=-Ddb2.jcc.propertiesFile=/u/db2Id/DB2_SSID -Xms128M -Xmx768M
   ```

9

# zRule Execution Server for z/OS embedded server

This chapter describes the process of how to use the embedded zRule Execution Server for z/OS (zRES) inside a COBOL or PL/I batch address space. This chapter also discusses the benefits of selecting the embedded zRES versus the stand-alone server.

The following topics are covered in this chapter:

## 9.1 Introduction to the embedded server

We are now firmly in the era of big data where data sets can be many gigabytes or even terabytes in size. The necessity to make decisions using this data means that moving the data to the rule engine can often be impractical, expensive, and time consuming. For example, when using the zRule Execution Server stand-alone to decide on 900,000 records, each record must be copied to and from the zRES for the decision to be made.

In this case, where the data sets are very large and the batch jobs processing them are very long, the zRule Execution Server embedded runtime may be a good choice of execution environment.

## 9.2 Move the decisions to the data

Since ODM was first available on z/OS, the HBRCON, HBRRULE, and HBRDISC API allowed batch clients to connect to a server and make decisions. The data that is required to make the decision is copied to the server and the rules are executed in the server. However, there is a cost to move the data to the server to execute the rules, and this cost increases as the size of the data grows.

The zRES embedded option changes this model by hosting the rule engine inside the COBOL or PL/I address space and makes the decisions locally. Therefore, the data is accessed locally (in situ) and no copying is necessary to make the decision.

Since ODM 8.6.0, the capability to launch the Rule engine within the COBOL or PL/I address space has been available. Applications still use the standard HBR API and make a simple configuration change to launch the rule engine locally, instead of connecting to a zRES stand-alone. This is illustrated in Figure 9-1.

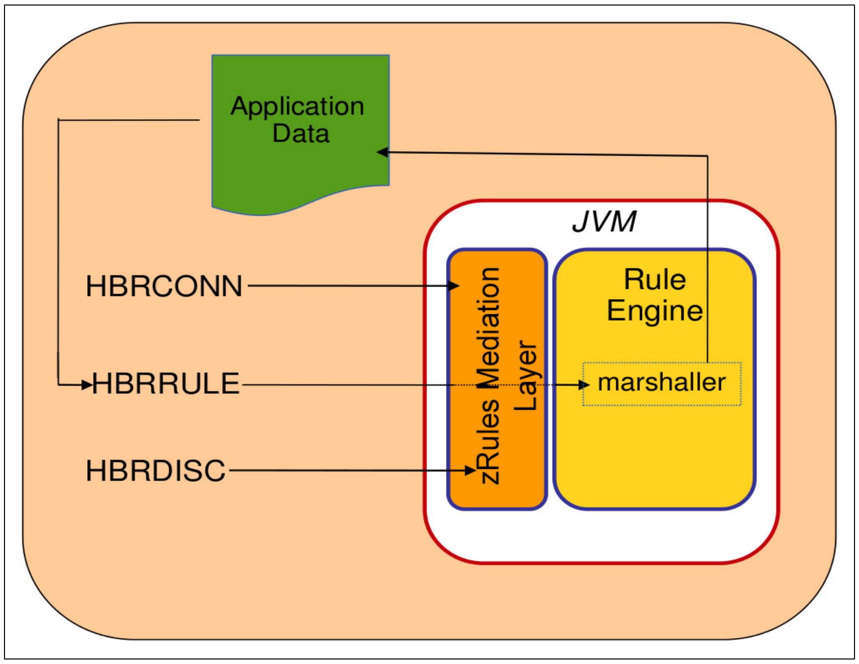


*Figure 9-1 The rule engine is hosted in the same address space as the application*

It is easy to configure an application to run the zRES embedded versus the zRES stand-alone. Therefore, it is important to understand the characteristics of your workload to ensure that it is best suited for the zRES embedded mode.

## 9.3 When to choose embedded over stand-alone server

The rule engine running in the zRES is a Java application, and as such undergoes a period of just-in-time (JIT) compilation after the zRES starts and also after the first rule execution. JIT compilation means that more CPU is used at the start of rule execution, while the JIT analyses execution paths and performs other optimizations to ensure that the code is more efficient in the long term.

A zRES stand-alone server is typically a long running server process that would not be restarted for weeks or even months. This means that JIT compilation occurs for a few minutes after zRES start, and then the code is fully optimized until the server is restarted, or new rulesets are deployed.

However, when using the embedded zRES, a new zRES server is started inside the batch job which means that each batch job undergoes this period of JIT optimization, which is very CPU intensive. The CPU used by the job also includes zRES server start, and ruleset loading.

Therefore, embedded mode brings the most benefit when the following conditions are true of the batch workload:

- Long running batch jobs: Those jobs that take a number of hours rather than a number of minutes are best. This is because the cost of the startup and JIT optimization is a smaller percentage of the overall batch execution.
- Small ruleset sizes: When the ruleset size is small (approximately 1000 rules or less), more time is spent getting to the zRES stand-alone than executing in it. Also, smaller rulesets take less time and CPU to complete JIT compilation.
- Large data size for evaluating rules: Large records take longer to copy to the zRES stand-alone for execution. Accessing them locally removes this cost.

Therefore, batch jobs that process many hundreds of thousands of large records using rulesets that are small are ideal candidates for zRES embedded.

**Note:** Performance of embedded zRES is stable after a 3 - 5-minute period of warmup consisting of 10’s of thousands of calls. Embedded is less costly in terms of CPU per call after a 4 - 5-minute period of warmup.

### 9.3.1 Other benefits of the embedded zRES

When your batch workload meets the criteria described in 9.3, “When to choose embedded over stand-alone server” on page 149, it is an ideal candidate for the zRES embedded mode.

Following are some other benefits that are also found when running the zRES in embedded mode:

- Memory constraints: The zRES embedded server only runs for the lifetime of the batch job and does not consume memory when not in use.
- Defined charge back to a job’s execution: The zRES embedded server runs as part of the batch job and CPU consumption occurs inside the batch job’s address space.

## 9.4 Configuring zRES embedded

The zRule Execution Server embedded mode is configured by using control statements, which are passed in to HBRUUPTI.

### 9.4.1 Batch control statements

ODM 8.6.0 introduces a new control statement command to create a BATCH execution environment. This control statement is used to create embedded mode working data sets, which the batch application uses to configure itself for zRES embedded mode.

When configuring the ++HBRHLQ++SHBRPARM(HBRCTRL), use the following batch control statements:

```
CREATE BATCH HBRWORKDS=RULES.WORK.EMBED
             HBRWORKPATH=/u/rules/workdirs/embed
```

Where, “HBRWORKDS” is the data set that is created and contains the JCL and parameter files, and “HBRWORKPATH” is the zFS location that is used by the rule engine.

### 9.4.2 Preparing the batch job for the embedded server

When the working data sets are created, it is possible to configure the batch job JCL to configure it for running the zRES embedded mode.

1. Submit ++HBRWORKDS++.SHBRJCL(HBRCRTI) to create the ++HBRWORKPATH++ for the zRES embedded mode
2. Configure the batch job JCL to add the following members to the STEPLIB:
    a. DD DISP=SHR,DSN=++HBRHLQ++.SHBRLOAD
    b. DD DISP=SHR,DSN=++HBRHLQ++.SHBRAUTH
3. Configure the batch job JCL to add the HBRENVPR DD statement, which points to the following data set concatenation:
    a. DD DISP=SHR,DSN=++HBRWORKDS++.SHBRPARM(HBREMBED)
    b. DD DISP=SHR,DSN=++HBRWORKDS++.SHBRPARM(HBRCMMN)
    c. DD DISP=SHR,DSN=++HBRWORKDS++.SHBRPARM(HBRPSIST)

**Note:** Ensure that the batch job has a large enough region size to accommodate the rule engine. The **`REGION`** parameter should be larger than the maximum JVM heap size specified in ++HBRWORKDS++.SHBRPARM(HBREMBED).

### 9.4.3 Executing zRES embedded

When the configuration is complete, submit your batch job to launch the application and execute rules locally using the zRES embedded mode. An example JCL is provided in ++HBRWORKDS++.SHBRJCL(HBRMINBE) to launch the Miniloan demo with an embedded zRES.

The output from the Miniloan demo running in a zRES embedded shows that the HBRA-CONN-SSID of the HBRA-CONN-AREA is set to the value “EMBD”. This signifies that the rule execution occurred locally in the zRES embedded.

### 9.4.4 Troubleshooting

The embedded zRES writes logs in a similar manner to the zRES stand-alone. The only major difference is that the embedded zRES does not include the HBRPRINT job output element. Therefore, the API completion and reason codes should be used to ensure the zRES embedded has started successfully.

**Note:** The zRES embedded returns HBR-RC-EMBED-JVM-ERR (3031) on the HBRCONN call when the embedded rule engine cannot be started locally.

When up and running, the zRES embedded writes log files if the HBRTRACELEVEL in ++HBRWORKPATH++.SHBRPARM(HBRCMMN) is set to a value that allows this to happen, for instance, FINE or ALL. The log files are then in the following locations:

- The SYSOUT and SYSPRINT of the batch job
- The ++HBRWORKPATH++/logs directory on the zFS

## 9.5 Not connecting to a console

By default, all zRES environments connect to a RES Console to receive notifications of ruleset updates. However, there might be a requirement that during the batch execution the ruleset version must not change because all records must be processed by the same version of a rule application and ruleset. This is important to consider when determining your type of rule execution server that you want to use in your environment. If a batch application is configured to always call the latest rule application and ruleset version, the embedded engine is able to disable ruleset updates during the execution if configured properly. This is in contrast to zRule Execution Server stand-alone, which has the potential to receive a ruleset update during the execution of a batch job. In this scenario, the zRES stand-alone starts executing with the new version and therefore records could potentially be executed with two different versions of the rules.

To meet this requirement, the following property can be set to disable the zRES connecting to the console on start.

**Note:** Set HBRCONSOLECOM=NO in ++HBRWORKDS++.SHBRPARM(HBRCMMN)

The other benefit of setting this property is that no CPU time is used by the batch job to establish and maintain the connection, thus speeding up the start time and reducing the overall CPU consumption slightly.

**Important:** The following limitations are experienced when setting this property:

- No ruleset updates during the batch execution
- No ruleset statistics collected for this engine

10

# Configuring IBM CICS to work with Operational Decision Manager

This chapter describes zRule Execution Server for z/OS when it is run within the CICS JVM server. It considers the configuration of CICS and the zRule Execution Server for z/OS for this environment. It also considers the use of rule owning regions and application owning regions.

The following topics are covered in this chapter:

## 10.1 Configuring CICS to invoke a stand-alone zRule Execution Server for z/OS

A CICS region can be configured so that a CICS program can call zRule Execution Server for z/OS (zRES). The CICS region is known as an application-owning region (AOR). Currently, the supported versions of CICS are 3.2, 4.1, 4.2, 5.1, and 5.2, as shown in Figure 10-1. This section describes the required configuration to enable this feature. It assumes that you already have a working zRES.

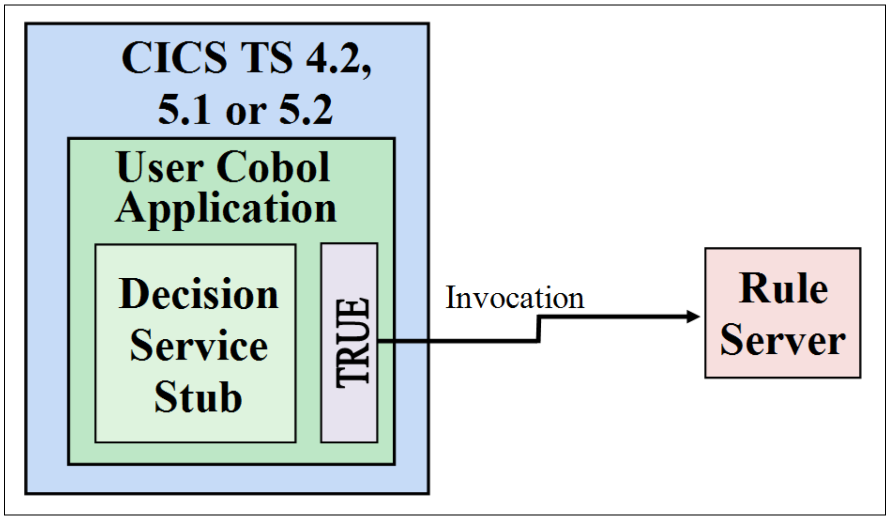


*Figure 10-1 CICS AOR and stand-alone server*

### 10.1.1 Creating working data sets for CICS

The following sections describe the steps to create the two working data sets for CICS:

- ++HBRWORKDS++.CICS.SHBRJCL
  The required JCL, specific to CICS rule execution
- ++HBRWORKDS++.CICS.SHBRPARM
  The runtime parameters

#### Define CICSLIST parameter in HBRINST

The CICSLIST parameter must be defined in HBRINST to run zRES with CICS. This value names the start group list that installs the resources that Operational Decision Manager requires. You can choose any valid name that you want.

Set this parameter in ++HBRHLQ++.SHBRPARM(HBRINST).

In this chapter, we use a value of “HBRLIST”.

#### Control statement for generating CICS working data sets

Edit ++HBRHLQ++.SHBRPARM(HBRCTRL) and add an entry to define the CICS working data set (substituting the ++variables++ for your own values):

```
CREATE CICS HBRWORKDS=++HBRWORKDS++.CICS
   CICSHLQ=++CICSHLQ++
   CICSCSDDSN=++CICSCSDDSN++
```

For explanations of the ++variables++ used in the preceding example, see Appendix B, “Configuring runtime values by using variables defined in HBRINST” on page 231

The HBRCTRL data set member is used by the HBRUUPTI job in the next step.

### Submit the HBRUUPTI job

After you defined the parameter and added the control statement to HBRCTRL, submit the following job to create the CICS working data set:

```
++HBRHLQ++.SHBRJCL(HBRUUPTI)
```

The CICS working data sets are created:

- ++HBRWORKDS++.CICS.SHBRPARM
- ++HBRWORKDS++.CICS.SHBRJCL

## 10.1.2 Defining the required resources

The resources that are required for CICS are defined by the JCL job:

```
++HBRWORKDS++.CICS.SHBRJCL(HBRCSD)
```

Submit this job to create the resources in a group named “HBRGROUP”.

## 10.1.3 Updating the GRPLIST parameter

After defining the resources, add the list name that you chose earlier (in “Define CICSLIST parameter in HBRINST” on page 154) to the **GRPLIST** parameter in the CICS system initialization table:

```
GRPLIST=(DFHLIST,HBRLIST)
```

## 10.1.4 Updating the CICS JCL

Modify the CICS region JCL to include the following changes, which allow access to the zRES.

### DFHRPL

In the CICS program library, `DFHRPL`, add the SHBRCICS PDS to the DFHRPL section:

```
//        DD DSN=++HBRHLQ++.SHBRCICS,DISP=SHR
```

### Passing the runtime variables to the CICS region

Adding the following to the CICS region JCL allows the client application access to the zRES parameters, including such things as its subsystem identifier (SSID).

```
//HBRENVPR DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRCICSZ)
//         DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRCMMN)
```

### Scenario for installation verification

If you plan to use the installation verification procedure (MiniloanDemo) to test the zRES on the CICS JVM server, add the following line in the runtime variables section:

```
//SCENARIO DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRSCEN)
```

After the configuration is complete and you restarted the region, this line provides the Miniloan sample with the scenario data that it requires. This sample application can be used to verify that the rule engine is connected and working.

### 10.1.5 Starting zRES and CICS

Start zRES and CICS now (or restart them if CICS is already running). To start zRES, use the following command:

```
START ++HBRSSID++MSTR
```

### 10.1.6 Installing HBRGROUP

Install the HBRGROUP resources to CICS by running the following command in CICS:

```
CEDA INSTALL GROUP(HBRGROUP)
```

### 10.1.7 Testing the configuration

The configuration can be tested by using the HBRC transaction. This transaction enables CICS to call zRES. The return code that is shown in Table 10-1 indicates the success of the transaction.

*Table 10-1 Return codes*

| Code | Meaning |
|---|---|
| GBRZC9000 | An error has occurred when executing the HBRC transaction. |
| GBRZC9001 | CICS has connected to zRES. |
| GBRZC9002 | CICS has disconnected from zRES. |
| GBRZC9003 | The CICS region is already connected to zRES. |

### 10.1.8 Automatically connecting CICS to a running zRES instance

This optional step means that it is not necessary to run the HBRC transaction to connect to a running zRES. The zRES must be started before the CICS region. Otherwise, you need to connect the CICS region by manually running HBRC.

There are two ways to automatically connect CICS to the running zRES instance:

- If you do not have a program list table defined, add the following parameter to the CICS system initialization table:

  ```
  PLTPI=HB
  ```

- If you have a program list table defined and specified in your CICS system initialization table, add the HBRCCON program to the list by using this statement:

  ```
  DFHPLT TYPE=ENTRY,PROGRAM=HBRCCON
  ```

### 10.1.9 Deploying and running the installation verification program

**Note:** To run the installation verification procedure (IVP), you must have specified the SCENARIO DD statement when editing the CICS start JCL (see 10.2.7, “Scenario for installation verification” on page 161).

Perform the following steps to run the IVP:

1. Deploy the RuleApp to the persistence layer by submitting the HBRDPLOY job in your `++HBRWORKDS++.CICS.SHBRJCL` CICS working data set.
2. After you deploy the RuleApp, go back to the CICS region and run the CICS transaction MINI. Your region then displays the output that is shown in Example 10-1.

*Example 10-1 Output from the Miniloan Demo IVP*

```
MINICICS--msg-The yearly income is lower than the basic request
MINICICS --Loan customer 0000000006
MINICICS --about to call # Execution Server
MINICICS -- Rule executed in-JVMS
MINICICS--name-Michelle             loan amount-0001000100-approved-F
MINICICS--msg-The loan cannot exceed 1000000
MINICICS --Disconnect from zRule Execution Server
MINICICS --SUCCESSFUL COMPLETION of demo
MINICICS--name-John                 loan amount-0000250000-approved-F
MINICICS--msg-The age exceeds the maximum.
MINICICS --Loan customer 0000000003
MINICICS --about to call zRule Execution Server
MINICICS -- Rule executed in-JVMS
MINICICS--name-Sarah                loan amount-0000500000-approved-F
MINICICS--msg-Credit score below 200
MINICICS --Loan customer 0000000004
MINICICS --about to call zRule Execution Server
MINICICS -- Rule executed in-JVMS
MINICICS--name-Andy                 loan amount-0000500000-approved-F
MINICICS--msg-Too big Debt-To-Income ratio
MINICICS --Loan customer 00000000 5
MINICICS --about to call zRule Execution Server
MINICICS -- Rule executed in-JVMS
MINICICS--name-David                loan amount-0000250000-approved-F
```

The setup of a CICS client application connected to a zRES server is complete.

## 10.2 Configuring zRES to run in a CICS JVM server

Another runtime feature of Operational Decision Manager is the addition of a zRES that runs within the CICS JVM server on CICS V4.2 and higher. The setup is similar for these versions of CICS, although the version of supported Java that is required to operate the CICS JVM servers within the region differs depending on the CICS version. This section describes the setup of a server instance, which is called a CICS *rule-owning region* (ROR) and is shown in Figure 10-2 on page 158. The instructions assume that the more general parameters in HBRINST are already defined, for example, from having configured a zRES.

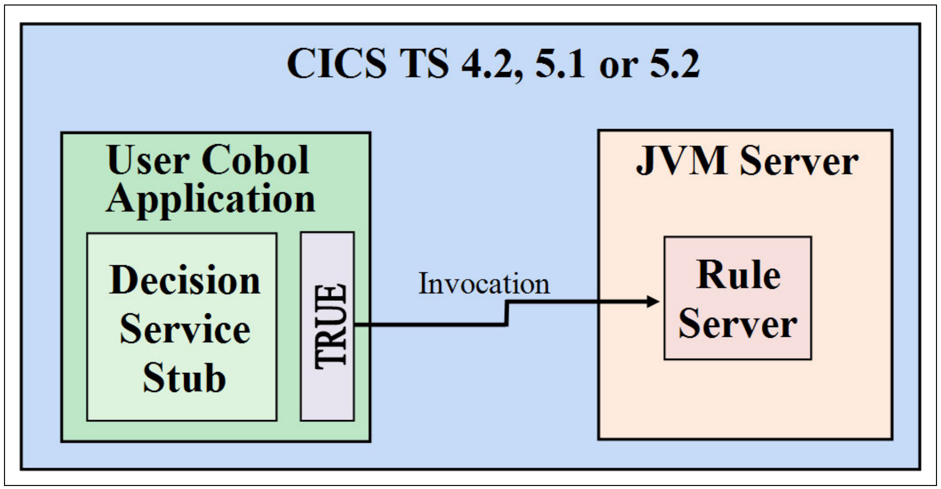


*Figure 10-2 CICS COBOL application and CICS JVM server*

## 10.2.1 Creating working data sets for CICS

The following sections describe the steps to create the two working data sets for CICS:

- ++HBRWORKDS++.CICS.SHBRJCL
  The required JCL, specific to CICS rule execution
- ++HBRWORKDS++.CICS.SHBRPARM
  The runtime parameters

### Define required parameters in HBRINST

You must update the ++HBRHLQ++.SHBRPARM(HBRINST) member to specify values for the following parameters:

- `CICSLIST`
  This value names the start group list that installs the resources that Operational Decision Manager requires. You can choose any valid name that you want.
- `JDBCPLAN`
  The plan that is used for JDBC connections in CICS.
- `HBRJAVAHOME`
  Specify a version of Java that is compatible with your CICS installation:
  - The CICS 4.2 environment supports Java 6.0.1 - 64 bit only.
  - The CICS 5.1 environment supports Java 7.0 - 64 bit only.
  - The CICS 5.2 environment supports Java 7.0.1 - 64 bit only.

### Control statement for generating CICS working data sets

Edit ++HBRHLQ++.SHBRPARM(HBRCTRL) and add an entry to define the CICS working data set (substituting the ++variables++ for your own values):

```
CREATE CICS HBRWORKDS=++HBRWORKDS++.CICS
   CICSHLQ=++CICSHLQ++
   CICSCSDDSN=++CICSCSDDSN++
   HBRWORKPATH=/u/cics/REGION1
```

For explanations of the ++variables++ used in the preceding example, see Appendix B, “Configuring runtime values by using variables defined in HBRINST” on page 231.

The HBRCTRL data set member is used by the HBRUUPTI job in the next step.

In Operational Decision Manager V8.6, CICSWORKPATH was removed from the control statement. If the environment that the zRule Execution Server is being configured requires a separate UNIX System Services path for CICS related files, add a variable in the control statement for called HBRWORKPATH to create the new workpath.

#### Submit the HBRUUPTI job

After you defined the parameters and added the control statement to HBRCTRL, submit the following job to create the CICS working data set:

```
++HBRHLQ++.SHBRJCL(HBRUUPTI)
```

The CICS working data sets are created:

- ++HBRWORKDS++.CICS.SHBRPARM
- ++HBRWORKDS++.CICS.SHBRJCL

### 10.2.2 Submitting jobs within the SHBRJCL working data set

Within the ++HBRWORKDS++.CICS.SHBRJCL data set that was created as a result of the last step, are a number of JCL scripts that need to be run to complete configuration of a Rule Execution Server in a CICS JVM.

#### Creating the working directories

The HBRCRTI job creates the CICS working path directories within UNIX System Services with the required configuration pieces, and a separate directory for the CICS JVM server logs.

Submit the HBRCRTI job.

#### Creating the JVM profile

The HBRCJVMP job creates a JVM profile for the CICS region, within the `++HBRWORKPATH++` directory (which is created in “Creating the working directories”).

Submit the HBRCJVMP job, which creates the JVM profile. This must be copied to the JVMPROFILEDIR of the CICS JVM servers.

#### Defining the CICS resources

Next, you define the CICS resources that are required by the server. To do this, submit these two jobs:

- HBRCSD
  This job defines the resources that are required by CICS regardless of where the rule engine is running.
- HBRCSDJ
  This job defines the extra resources that are required when the rule engine runs within a CICS JVM.

## 10.2.3 Adding ++CICSLIST++ to the CICS system initialization table

Having defined the resources in the last step, add the value that you chose for ++CICSLIST++ (in "Define required parameters in HBRINST" on page 158) to the CICS system initialization table that is specified by the **GRPLIST** parameter:

```
GRPLIST=(DFHLIST,HBRLIST)
```

## 10.2.4 Setting the JVMPROFILEDIR

If you do not already have a JVM profile directory defined, you need to set the default JVM profile to point at the working directory that you created for CICS in ++HBRWORKPATH++.

In the CICS system initialization table, create the `JVMPROFILEDIR` variable. It needs to point to the CICS working directory:

```
JVMPROFILEDIR=++HBRWORKPATH++
```

If you use a `JVMPROFILEDIR` other than ++HBRWORKPATH++, you must copy the profile that was created in the last step (10.2.3, "Adding ++CICSLIST++ to the CICS system initialization table" on page 160) to the correct `JVMPROFILEDIR` directory.

## 10.2.5 Changing the CICS region JCL

The CICS region JCL must be modified to include the lines that call the zRES on the CICS JVM server.

### DFHRPL

In the CICS program library, `DFHRPL`, add the SHBRCICS PDS to the DFHRPL section:

```
//        DD DSN=++HBRHLQ++.SHBRCICS,DISP=SHR
```

### Passing the runtime variables to the CICS region

Adding the following to the CICS region JCL allows the client application access to the zRES parameters, including such things as its SSID.

```
//HBRENVPR  DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRCICSJ)
//          DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRCMMN)
//          DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRCICSD)
```

The HBRCICSD member in ++HBRWORKDS++.CICS.SHBRPARM sets up the database environment.

## 10.2.6 Providing a console for the zRule Execution Server

The zRule Execution Server for z/OS within a CICS JVM requires a console to allow deployment and management of rule applications just as a standard zRES server does. This console is defined in the same way as for other configurations by adding a control statement to ++HBRHLQ++.SHBRPARM(HBRCTRL) with a mode of CONSOLE. This creates working data sets that start a zRES in console mode.

### 10.2.7 Scenario for installation verification

If you plan to use the IVP to test the zRES on the CICS JVM server, add the following line in the runtime variables section:

```
//SCENARIO DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRSCEN)
```

After the configuration is complete and the region is restarted later, this line provides the Miniloan sample with the scenario data it requires. This sample application can be used to verify that the rule engine is connected and working.

### 10.2.8 Security for the zRES on CICS JVM server

For the security setup for the zRES on the CICS JVM server, perform the steps in 8.2.3, “Securing the zRule Execution Server for z/OS for z/OS resources” on page 137. You perform the same steps for all zRES servers. The CICS region’s user for the started task must be granted access to the connect security so that it can connect to the zRES instance for rule execution.

### 10.2.9 CEDA installation of HBRGROUP resources

After you start the CICS region, you must install the resources that were defined earlier. Run the following command in CICS:

```
CEDA INSTALL GROUP(HBRGROUP)
```

### 10.2.10 Database connect for the CICS region

Run the following command to connect the database to the CICS region of the zRES:

```
CEMT INQUIRE DB2CONN
```

Then, change the CONNECTST property from `Notconnected` to `Connected`.

**Tip:** You can also use the **`DB2CONN=YES SIT`** parameter to perform this connection automatically on CICS start.

### 10.2.11 Initializing the zRES in the CICS JVM server

After CICS starts, initialize the zRES in the CICS JVM server by using the CICS transaction HBRC. This transaction sets up storage in the CICS JVM for connection to zRES and checks and initializes the JVM. If successful, this transaction returns the following message:

```
GBRZC9001I RC=0000
```

If the connection is unsuccessful, it returns `GBRZC9001E RC=`*`XXXX`*, where *`XXXX`* is the return code message.

For more information about these return codes, see the *Completion codes* topic in the Decision Server for z/OS V8.7 product documentation:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.ref/html/reasoncodes/html/codes_zres.html

### 10.2.12 Deploying the installation verification program

**Note:** To run the IVP, you must have specified the SCENARIO DD statement when editing the CICS start JCL (see 10.2.7, “Scenario for installation verification” on page 161).

Perform the following steps to run the IVP:

1. Deploy the RuleApp to the persistence layer by running the HBRDPLOY job in your `++HBRWORKDS++.CICS.SHBRJCL` CICS working data set.
2. After you deploy the RuleApp, go back to the CICS region and run the CICS transaction MINI. Your region then displays the output that is shown in Example 10-2.

*Example 10-2 MINI output*

```
MINICICS--msg-The yearly income is lower than the basic request
MINICICS --Loan customer 0000000006
MINICICS --about to call # Execution Server
MINICICS -- Rule executed in-JVMS
MINICICS--name-Michelle           loan amount-0001000100-approved-F
MINICICS--msg-The loan cannot exceed 1000000
MINICICS --Disconnect from zRule Execution Server
MINICICS --SUCCESSFUL COMPLETION of demo
MINICICS--name-John               loan amount-0000250000-approved-F
MINICICS--msg-The age exceeds the maximum.
MINICICS --Loan customer 0000000003
MINICICS --about to call zRule Execution Server
MINICICS -- Rule executed in-JVMS
MINICICS--name-Sarah              loan amount-0000500000-approved-F
MINICICS--msg-Credit score below 200
MINICICS --Loan customer 0000000004
MINICICS --about to call zRule Execution Server
MINICICS -- Rule executed in-JVMS
MINICICS--name-Andy               loan amount-0000500000-approved-F
MINICICS--msg-Too big Debt-To-Income ratio
MINICICS --Loan customer 00000000 5
MINICICS --about to call zRule Execution Server
MINICICS -- Rule executed in-JVMS
MINICICS--name-David              loan
amount-0000250000-approved-F
```

The setup of the zRES on the CICS JVM server is complete.

## 10.3 Working with multiple CICS JVM servers

Rules execution can be run through multiple separate CICS systems on the same LPAR, with the execution units (XUs) being deployed to each CICS system. This topology is displayed in Figure 10-3.

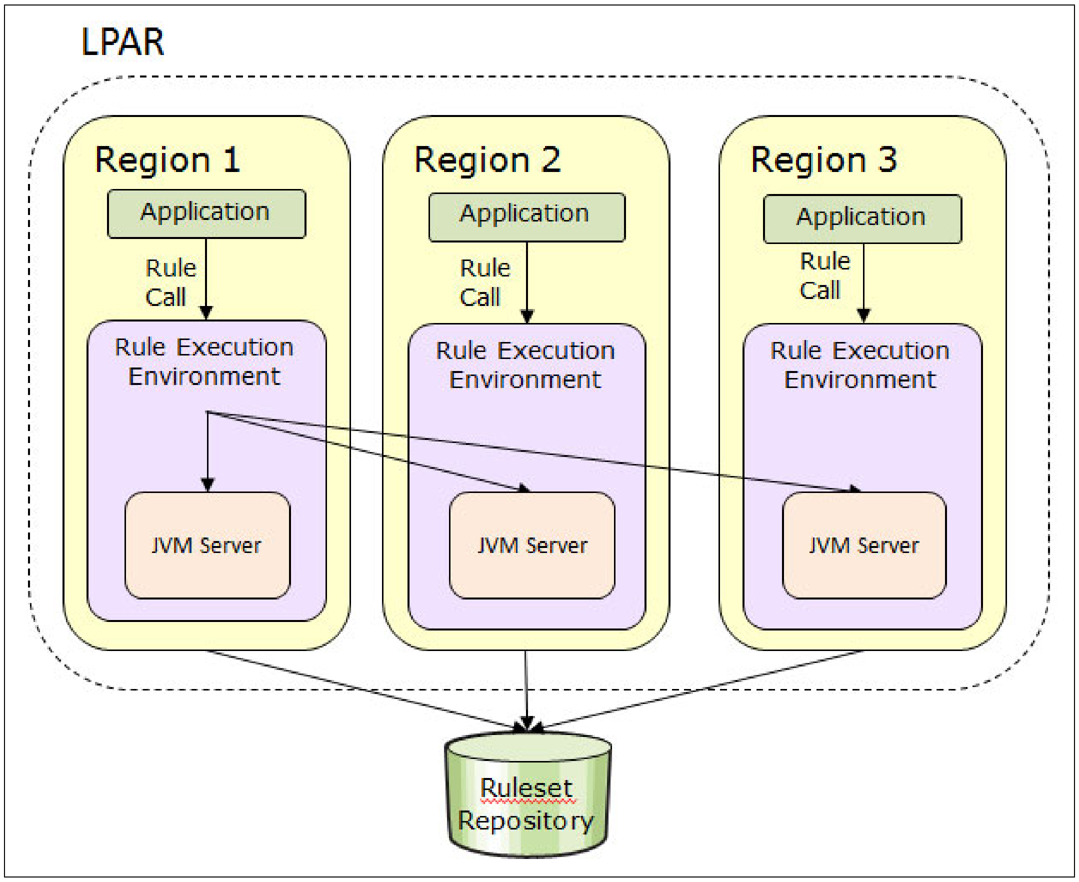


*Figure 10-3 Multiple separate CICS JVM servers*

Using this method allows different CICS systems to simultaneously access the same set of rules. They might be running different applications or have different uses that require them to be separate from each other.

To set up multiple CICS JVM servers, first set up a single server by using the method that is described in 10.2, “Configuring zRES to run in a CICS JVM server” on page 157.

Additional information is in the *Configuring a CICS rule-owning region to execute rules in a CICS JVM server* topic in IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/tsk_ds_config_cics_jvm.html

The working directory that is created as part of this process can be used by all servers in the IBM CICSPlex®. All servers can also use the default CICS JVM profile in that working directory.

### 10.3.1 Using the same JVM profile and working directory

If you want to use the same working directory and JVM profile for all the CICS regions, this is the default behavior.

If you want to run the Installation Verification Sample on each region, you also need to add the runtime variables DD statement to the CICS region JCL (see "Scenario for installation verification" on page 155.

## 10.4 Rule-owning regions and application-owning regions

In the previous sections of this chapter, the configurations described have always involved an application making a rule call to a rule execution environment on the same CICS region. It is however possible to separate the client application program from the rule execution environment by placing them in different regions:

- One or more regions contain the rule execution environment, hosted inside their JVM server. These are called *rule-owning regions* (RORs).
- One or more regions contain the client application. These are called *application-owning regions* (AORs) and execute their decisions remotely on an ROR.

This is illustrated in Figure 10-4.

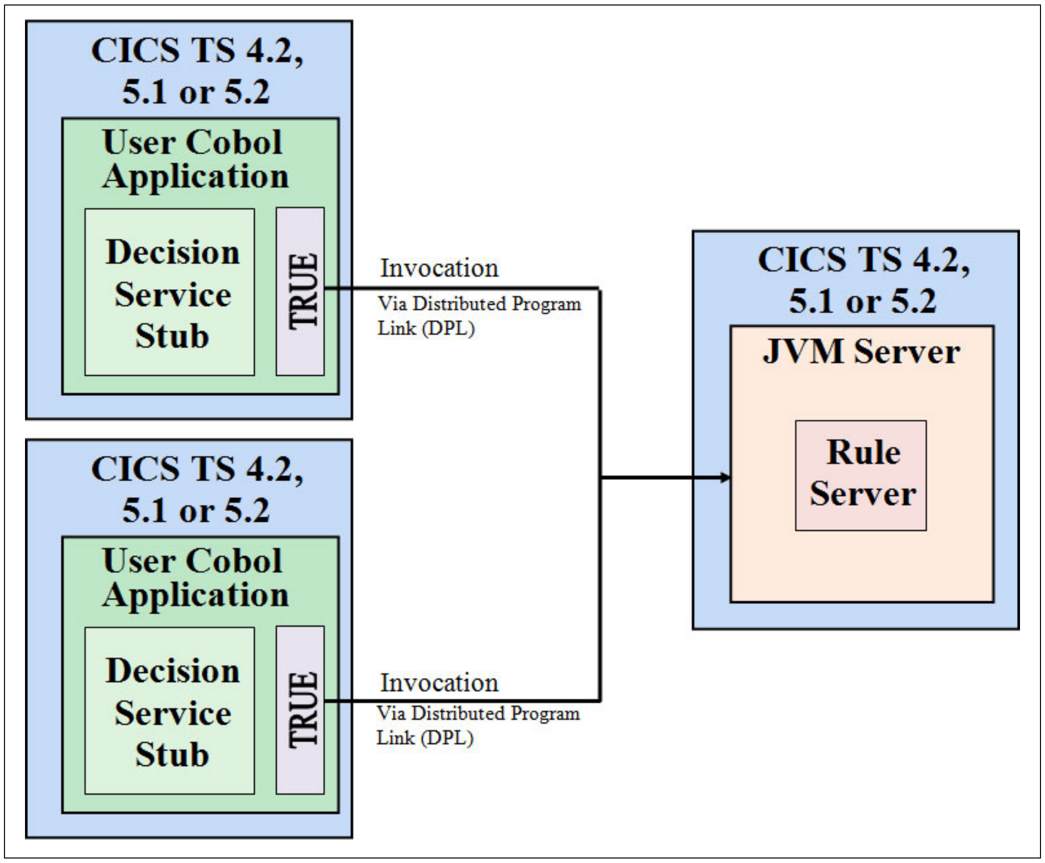


*Figure 10-4 CICS TS AORs executing decisions remotely on an ROR*

In Figure 10-4 on page 164, CICS TS AORs can communicate with the ROR using a distributed program link (DPL) or by using CICSPlex SM workload management (WLM). The use of DPL or WLM allows the decision request to be routed dynamically to the ROR. This provides a highly available and work load managed solution when two or more RORs are used.

The next section details how this architecture also provides a cost effective way for the COBOL and PL/I applications running in the AORs to execute business decisions using Operational Decision Manager for z/OS running in the ROR.

## 10.4.1 Cost effectiveness

The cost of Operational Decision Manager for z/OS is based on the size of the LPARs that it is deployed into regardless of whether Decision Server is running in a zRule Execution Server for z/OS, WebSphere Application Server for z/OS, or CICS Transaction Server for z/OS. The pricing of Operational Decision Manager for z/OS is not affected by the CICS TS pricing model (Value Unit Edition or Monthly License Charge). The products are sold and priced independently.

However, IBM CICS Transaction Server for z/OS Value Unit Edition offers a unique way to contain the cost of Operational Decision Manager for z/OS for CICS TS applications running in the AORs through two means:

- The CICS TS rule-owning region allows the isolation of the costs of Operational Decision Manager into a single LPAR:
  - There is no cost for the Operational Decision Manager for z/OS client libraries running on the AOR. The client libraries provide the API required to execute rules on the ROR.
  - Multiple AORs can route decision requests to Operational Decision Manager for z/OS running in the ROR.
- z/NALC LPARs are separate from z/OS LPARs and are often smaller in size. Therefore, the cost of Operational Decision Manager for z/OS is reduced.

Operational Decision Manager for z/OS also requires a DB2 database to store the runtime artifacts and also to support runtime warehousing features. Another benefit of running Operational Decision Manager for z/OS inside a z/NALC LPAR is that DB2 for z/OS Value Unit Edition can be used to provide this persistence layer, further reducing the cost of implementing Decision Management on z/OS.

Following are the steps to configure an application-owning region.

## 10.4.2 Create working data sets for the AOR region

Follow the instructions in 10.1.1, “Creating working data sets for CICS” on page 154 to create working data sets for the region.

This results in two data sets:

- ++HBRWORKDS++.CICS.SHBRJCL
  The required JCL, specific to CICS rule execution.
- ++HBRWORKDS++.CICS.SHBRPARM
  The runtime parameters.

### Installation on a different MVS image or sysplex

If your Operational Decision Manager installation is on a different IBM MVS™ image or sysplex, after creating the working data sets you need to copy these and a few other data sets to the MVS image of the application-owning region that you are configuring.

Copy the ++HBRHLQ++.SHBRCICS data set and the following members:

```
++HBRWORKDS++.CICS.SHBRJCL(HBRCSD)
++HBRWORKDS++.CICS.SHBRPARM(HBRCICSZ)
++HBRWORKDS++.CICS.SHBRPARM(HBRCMMN)
++HBRWORKDS++.CICS.SHBRPARM(HBRSCEN)
```

## 10.4.3 Define the required CICS connection resources

For the AOR and ROR to communicate with each other, some required resources must be defined. These are referenced by the program definition.

For more information about remote connections in CICS, see the IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.intercommunication.doc/topics/dfht12b.html

### CONNECTION resource

If the two regions are to communicate by using intersystem communication (ISC) or multiregion operation (MRO), a CONNECTION definition is required. The name of this definition is used in the next step as the REMOTESYSTEM name.

### SESSION resource

In addition to a CONNECTION definition, if you are using ISC or MRO as the connection between the regions, you also require a SESSION resource to define properties of the connection. This resource is not referenced directly by the program definition.

### IPCONN resource

If the two regions are to communicate by using TCP/IP, an IPCONN definition is required. The name of this definition is used in the next step as the REMOTESYSTEM name.

## 10.4.4 Customize the HBRCSD JCL to use a remote server program

The default configuration of the resources in `++HBRWORKDS++.CICS.SHBRJCL(HBRCSD)` is for a CICS client application calling a rule execution environment that is on the same region. Within the job, there is a definition template for remote rule execution, which is commented out:

1. Comment out the existing definition of HBRCJVMS in HBRCSD.
2. Uncomment the remote definition that follows in the JCL.
3. In the program definition, enter a value for the remote system. This value should match an existing IPCONN or CONNECTION definition that defines the remote CICS system.

Example 10-3 on page 167, HBRCSD to use a rule owning region, shows how the JCL should look after being edited, although the value MYIPCONN would be exchanged for the name of your regions IPCONN or CONNECTION definition.

*Example 10-3 HBRCSD to use a rule owning region*

```
*  DEFINE PROGRAM(HBRCJVMS)       GROUP(HBRGROUP)
*     LANGUAGE(ASSEMBLER)      RELOAD(NO)            EXECKEY(USER)
*     RESIDENT(NO)             USAGE(NORMAL)         USELPACOPY(NO)
*     STATUS(ENABLED)          CEDF(YES)             DATALOCATION(ANY)
*     CONCURRENCY(THREADSAFE)
*     DESCRIPTION(IBM Decision Server for zOS)

 DEFINE PROGRAM(HBRCJVMS)       GROUP(HBRGROUP)
   STATUS(ENABLED)            DYNAMIC(NO)
   REMOTESYSTEM(MYIPCONN) REMOTENAME(HBRCJVMS)
   DESCRIPTION(IBM Decision Server for zOS)
```

## 10.4.5 Define the required resources

The resources required for CICS are defined by the JCL job edited in the previous step:

```
++HBRWORKDS++.CICS.SHBRJCL(HBRCSD)
```

Submit this job to create the resources.

The resources are defined in a group named HBRGROUP.

## 10.4.6 Edit the HBRCICSZ file to specify a remote target Rule Execution Server

Edit the `++HBRWORKDS++.CICS`.SHBRPARM(HBRCICSZ) member. Change the HBRTARGETRES variable to RCICSJVM. This specifies the client application to use a remote execution configuration.

## 10.4.7 Updating the GRPLIST parameter

After defining the resources, add the list name that you chose earlier (in “Define CICSLIST parameter in HBRINST” on page 154) to the **`GRPLIST`** parameter in the CICS system initialization table:

```
GRPLIST=(CICSHTAP,HBRLIST)
```

## 10.4.8 Updating the CICS JCL

Modify the CICS region JCL to include the following changes, which allow access to the zRES.

### DFHRPL

In the CICS program library, `DFHRPL`, add the SHBRCICS PDS to the DFHRPL section:

```
//        DD DSN=++HBRHLQ++.SHBRCICS,DISP=SHR
```

### Passing the runtime variables to the CICS region

Adding the following to the CICS region JCL allows the client application access to the zRES parameters, including such things as its subsystem identifier (SSID).

```
//HBRENVPR DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRCICSZ)
//         DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRCMMN)
```

### Scenario for installation verification

If you plan to use the IVP to test the zRES on the CICS JVM server, add the following line in the runtime variables section:

```
//SCENARIO DD DISP=SHR,DSN=++HBRWORKDS++.CICS.SHBRPARM(HBRSCEN)
```

After the configuration is complete and the region is restarted later, this line provides the Miniloan sample with the scenario data that it requires. This sample application can be used to verify that the rule engine is connected and working.

## 10.4.9 Installing HBRGROUP

Start CICS and install the HBRGROUP resources to CICS by running the following command in CICS:

```
CEDA INSTALL GROUP(HBRGROUP)
```

## 10.4.10 Testing the configuration

The configuration can be tested by using the HBRC transaction. This transaction enables CICS to call zRES. The return code that is shown in Table 10-1 on page 156 indicates the success of the transaction.

Start CICS and issue the HBRC transaction.

To further verify the configuration, you can use the Installation Verification Sample (see 10.1.9, "Deploying and running the installation verification program" on page 157).

## 10.4.11 Further information about configuring an AOR and ROR

Further information about the steps that are required to configure an AOR and ROR can be found in IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/tsk_ds_config_topol_2_3.html

11

# Configuring IBM IMS to work with Operational Decision Manager

This chapter describes the use of Operational Decision Manager for z/OS with IMS.

The following topics are covered in this chapter:

## 11.1 IMS and Operational Decision Manager

Using Operational Decision Manager from IMS allows the rules that influence the business logic to be kept outside the IMS environment. Encapsulating the rules from these decision points outside of the IMS application decouples the rules from the IMS applications. This allows the business rules to be more reactive to changes without having to modify the IMS applications. IMS can call Operational Decision Manager from programs running in the message processing regions (MPR), Batch Message Processing (BMP), or Data Language/I (DL/I) programs. IMS and the zRule Execution Server for z/OS (zRES) must reside on the same logical partition (LPAR).

IMS uses the same API calls that are used by batch and CICS programs:

- HBRCONN: To connect to the server group
- HBRRULE: To run rules
- HBRDISC: to disconnect from the server group

However, these API calls use IMS dedicated stubs that are contained in a separate library, as shown in Figure 11-1.

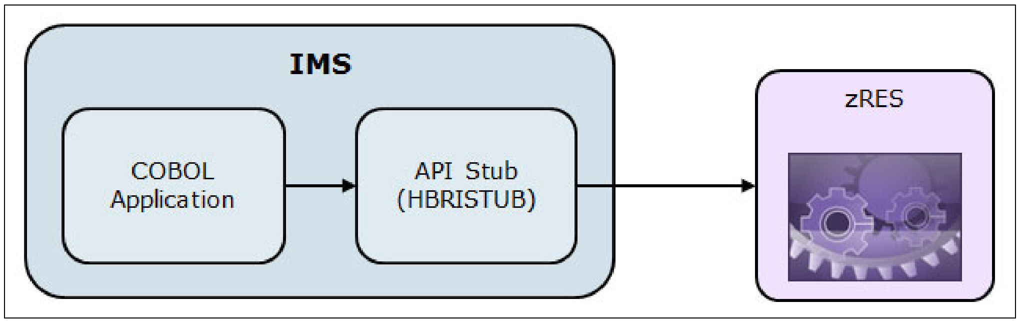


*Figure 11-1 IMS calling into zRES*

## 11.2 Configuration

This section describes the required configuration for the different types of programs within IMS.

To resolve the API calls HBRCONN, HBRRULE, and HBRDISC, the IMS program needs to be link-edited with the HBRISTUB module. To do this include the following link-edit step when binding the client program:

```
INCLUDE HBRLIB (HBRISTUB)
```

Three configuration parameters relate to IMS:

- IMSHLQ: The high-level qualifier (HLQ) of the IMS installation
- IMSREGID: The region ID of the IMS region to be used
- IMSREGHLQ: The HLQ of the IMS region to be used

These three configuration parameters are used in the creation of the provided sample programs (HBRMINI and HBRMINIT). Although it is useful to set them up, it is not necessary. zRES can be used without them being set up, if the other configuration steps are followed.

## 11.2.1 BMP and DL/I

There is no additional setup required for BMP and DL/I programs to call into the zRES other than including the ODM HBR API calls.

## 11.2.2 Message processing region

When running your rule decisions in this environment, configure the IMS system so that the Operational Decision Manager IMS preinitialization module receives control before the MPR-dependent regions are initialized. This preinitialization routine is called “HBRIPREI”.

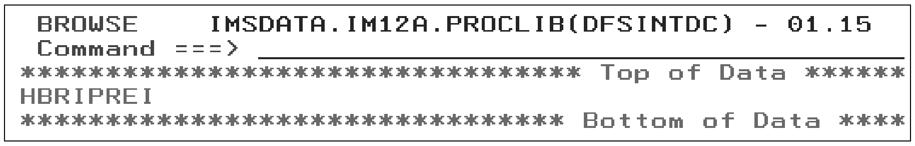
This load module is found in the Operational Decision Manager SHBRLOAD library.

### The DFSINT member

Use the DFSINTxx member of the IMS PROCLIB data set to identify the preinitialization module.

Edit this member with the name of the preinitialization program. See Figure 11-2.

HBRIPREI is the preinitialization program.



```
 BROWSE     IMSDATA.IM12A.PROCLIB(DFSINTDC) - 01.15
 Command ===> ________________________________________
********************************** Top of Data ******
HBRIPREI
********************************* Bottom of Data ****
```

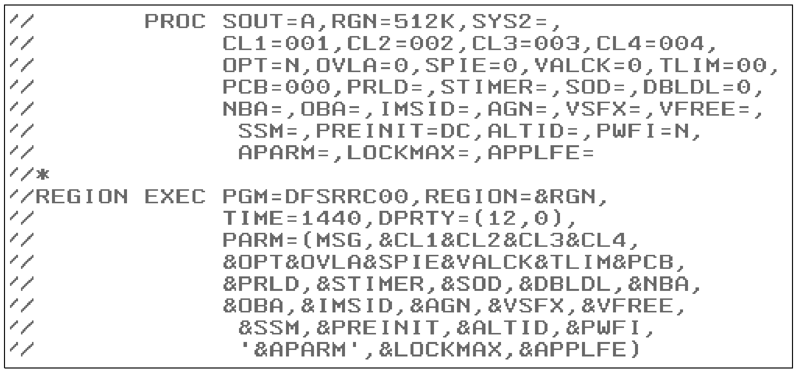
*Figure 11-2   DFSINITxx member*

### The DFSMPR procedure

This is the execution procedure that initiates an IMS message processing address space.

Edit the DFSMPR procedure’s **`PREINIT`** parameter to point to this DFSINTxx member: PREINIT=DC.

The extract in Figure 11-3 shows this setting:



```
//        PROC SOUT=A,RGN=512K,SYS2=,
//             CL1=001,CL2=002,CL3=003,CL4=004,
//             OPT=N,OVLA=0,SPIE=0,VALCK=0,TLIM=00,
//             PCB=000,PRLD=,STIMER=,SOD=,DBLDL=0,
//             NBA=,OBA=,IMSID=,AGN=,VSFX=,VFREE=,
//              SSM=,PREINIT=DC,ALTID=,PWFI=N,
//              APARM=,LOCKMAX=,APPLFE=
//*
//REGION EXEC PGM=DFSRRC00,REGION=&RGN,
//             TIME=1440,DPRTY=(12,0),
//             PARM=(MSG,&CL1&CL2&CL3&CL4,
//             &OPT&OVLA&SPIE&VALCK&TLIM&PCB,
//             &PRLD,&STIMER,&SOD,&DBLDL,&NBA,
//             &OBA,&IMSID,&AGN,&VSFX,&VFREE,
//              &SSM,&PREINIT,&ALTID,&PWFI,
//              '&APARM',&LOCKMAX,&APPLFE)
```

*Figure 11-3   DFSMPR execute procedure extract*

Message processing region (MPR) programs require extra setup to access zRES. In the message processing JCL, you need to add the following information to the IMS started task:

1. An HBRENVPR statement that gives the location of the data set member. This contains the details about the location of the rules environment:

   ```
   //HBRENVPR DD DISP=SHR,
   // DSN=++HBRWORKDS++.SHBRPARM(HBRBATCH)
   ```

2. Include the zRES load library in the STEPLIB and DFSESL statements:

   ```
   // DD DSN=++HBRHLQ++.SHBRLOAD,DISP=SHR
   ```

3. Restart your IMS Message processing region so that it uses the changes that you made to the JCL.

## 11.3 IMS and Rule Execution Server on WebSphere Application Server for z/OS

Further configuration steps are necessary to run rules on Rule Execution Server on WebSphere Application Server for z/OS. These steps are described in 13.5, “IMS and Rule Execution Server using WOLA on z/OS” on page 191.

12

# Liberty Application Server on IBM z/OS

This chapter introduces the *Liberty Application Server on z/OS* (Liberty on z/OS) and describes the Operational Decision Manager capabilities in this environment.

The following topics are covered in this chapter:

## 12.1 Introduction

Liberty on z/OS is an application server that has been written to use specific features of z/OS. It is ideal for users that want to host their web server applications on a lightweight server. It fulfills the following wish list:

- Lightweight: Uses a small footprint server.
- Composable: Server can be configured to only have the functions that the applications require.
- Simple: Single user can manage servers and applications in a configuration file.
- Flexible: Shares configuration across servers and hosts; deploys using compressed files.
- Dynamic: Does not restart the server for every configuration change.
- Extensible: Starts small then adds capability; adding your own runtime capability.

#### How lightweight

An installable .JAR file that provides all of the Java EE web profile in less than 54 MB.

The server starts in less than 3 seconds with a JSP/JDBC application installed.

The server configuration allows you to control which features are loaded into a given server instance at a fine-grained level, so you get exactly the function you want and no more.

Each feature is self-contained both from an installation and a runtime perspective. An important aspect of this is the configuration metadata, which includes a set of default values.

A major aspect of the simplicity of Liberty is the configuration files. Instead of dozens of separate XML files over many directories, you can configure a server in a single XML file.

## 12.2 Liberty on z/OS and Java

Liberty has been written to use the various reliability, availability, and serviceability capabilities of z/OS. In Liberty, there are specific features for each of these areas.

### 12.2.1 Reasons to use Java on z/OS

z/OS has specialty engines for running Java work rather than the general processor engines.

In addition, the Java virtual machine has been specially written to use the specifics of the z/OS hardware.

By moving workloads to z/OS, you can leverage the quality of service, management, and scalability inherent on z/OS.

### 12.2.2 Collocation

z/OS is often the platform of choice for the location of your data. Improving the proximity of your application to your data is desirable.

### 12.2.3 Management

z/OS has exceptionally good management systems for managing and controlling your workloads.

Tight integration with z/OS Workload Manager enables policy-based prioritization of batch and online workloads to gain maximum utilization of the resources.

### 12.2.4 Security

System Authorization Facility (SAF) provides a single point of security administration with a strong tradition of careful, controlled security management.

### 12.2.5 Transactions

Resource Recovery Services (RRS) is a sysplex-enabled transaction sync point coordinator facility integrated with z/OS. It excels at TX processing and recovery.

## 12.3 Operational Decision Manager running on Liberty on z/OS

In this section, we examine the capabilities of Operational Decision Manager that can be run in Liberty on z/OS.

### 12.3.1 Capabilities that can run in Liberty on z/OS

The following Operational Decision Manager capabilities can run in Liberty on z/OS:

- The Rule Execution Server console can be run on Liberty. This is an ideal use case because Liberty is small, lightweight, and fast to start. The benefits in running the console in Liberty provide a small footprint plus the speed advantage of deploying cross memory to DB2 by using a type 2 connection. This deployment occurs while using the advanced SAF capability of z/OS through a security provider such as Resource Access Control Facility (RACF).
- The hosted transparent decision service (HTDS) application is supported giving client application access to business rules via Web Services and RESTful interfaces. The Web Services Description Language (WSDL) file for the web service can be obtained by selecting SOAP when retrieving the HTDS description file. The Web Application Description File (WADL) for the Representational State Transfer (REST) service can be obtained by selecting REST when retrieving the HTDS description file.
- Testing and simulation capabilities. Operational Decision Manager testing and simulation capabilities supplied by the Decision Runner application can now be hosted in this environment.

## 12.4 Installation and configuration of Liberty

Liberty is currently not packaged with Operational Decision Manager, but is available as a separate package from WebSphere Application Server.

The UNIX System Services installation directory structure for Liberty looks something like this (Figure 12-1).

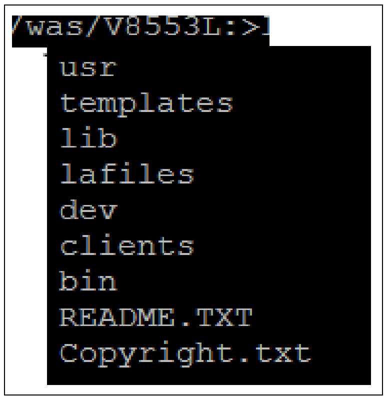

```
/was/V8553L:>l
  usr
  templates
  lib
  lafiles
  dev
  clients
  bin
  README.TXT
  Copyright.txt
```

*Figure 12-1 Liberty directory structure*

## 12.4.1 A couple of subdirectories of interest

Table 12-1 shows two of the subdirectories of the Liberty installation that are of interest.

*Table 12-1 Subdirectories of the Liberty installation*

| Subdirectory | Use |
|---|---|
| templates/ | Runtime customization templates and examples |
| templates/zos/procs/ | z/OS proc templates for starting a Liberty server |

The `README.TXT` file describes the actions that the server command can initiate.

## 12.4.2 Useful environment variables

Refer to the “Liberty Profile z/OS Quick Start Guide”, which is available at the following website:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102110

The “Quick Start” guide has a good overview and step-by-step instructions for creating a Liberty server from the UNIX shell.

Environment variables that are shown in Table 12-2 on page 177 need to be set in your environment before you start the process of defining a new application server.

Table 12-2 Environment variables of interest for Liberty

| Variable | Description |
|---|---|
| JAVA_HOME | Location of 64-bit Java 6 or Java 7 |
| WLP_USER_DIR | Location where the Server definitions reside |
| WLP_INSTALL_DIR | Root of WebSphere Liberty Profile |
| _BPXK_AUTOCVT=ON | Converts text files to the tagged code page for the file |

With the **`_BPXK_AUTOCVT=ON`** variable set, it is then possible to tag files with the appropriate code page. See Example 12-1.

*Example 12-1 Tagging files*

```
chtag -t -c iso8859-1 <filename>

ls -lT  shows tagged files

where <filename> is the name of the file to be tagged
```

## 12.4.3 Creating the server

After the previous variables are set, change the directory into the location given by the `WLP_INSTALL_DIR` variable and then change into the bin sub-directory.

Then, *create* a server by using the following command:

```
server create myServer
```

where “myServer” is the name of a new server

To *start* the server, use the command:

```
server start myServer
```

To *stop* the server, use the following command:

```
server stop myServer
```

To view the output from the server, use the following steps:

1. Change the directory to the location given by `WLP_INSTALL_DIR`.
2. Then, change to the `servers` subdirectory and you should find a directory called `myServer`.
3. Change to this directory and then the subsequent `logs` directory and view the `messages.log` file, which is an ASCII file.

**Note:** To allow the ASCII format file messages.log to be processed from USS, either:

- Tag the `messages.log` file, allowing the OS to see the file is an ASCII file

OR

- Use a tool such as **`viascii`**

## 12.4.4 The directory structure

After the "server create" command is executed, a directory structure beneath `WLP_USER_DIR` is created.

The directory structure might look like Figure 12-2 for a server called *myServer*.

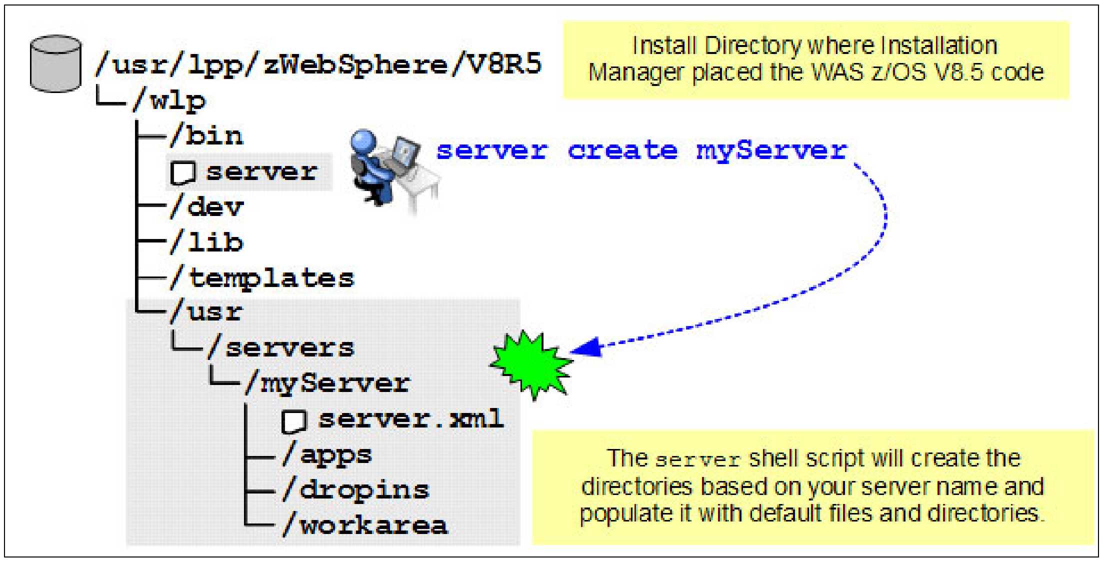

/usr/lpp/zWebSphere/V8R5
└/wlp
├/bin
server
├/dev
├/lib
├/templates
└/usr
└/servers
└/myServer
server.xml
├/apps
├/dropins
└/workarea
Install Directory where Installation Manager placed the WAS z/OS V8.5 code
server create myServer
The server shell script will create the directories based on your server name and populate it with default files and directories.

*Figure 12-2 Creation of myServer*

## 12.4.5 Liberty configuration files

Under the `myServer` directory (Figure 12-3 on page 179), there are a number of configuration files:

- `server.env`
- `jvm.options`
- `server.xml`

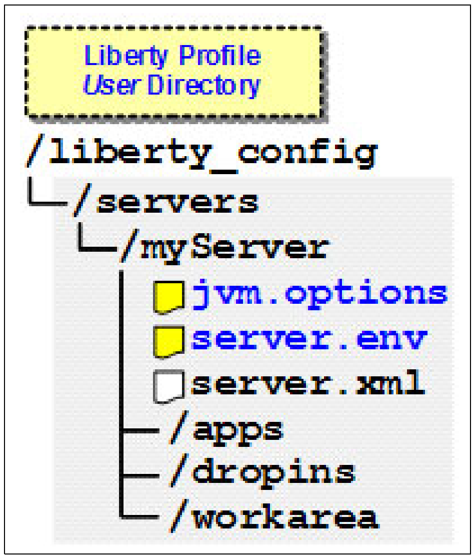


*Figure 12-3 Configuration files*

### server.env

This file gives the location of Java.

### jvm.options

As the name suggests, this file specifies anything that is related to the JVM, such as server port numbers.

### server.xml

This file contains the configurations settings for this Liberty server. The file gets replaced with the one tailored for the Operational Decision Manager's (ODMs) capability.

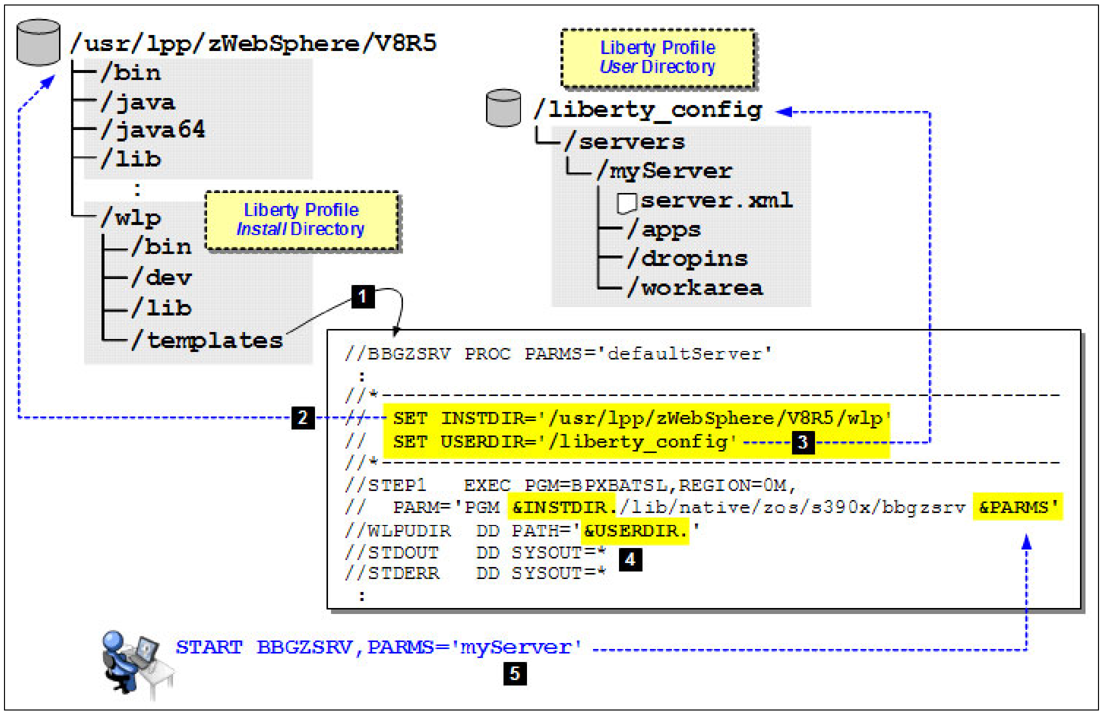
## 12.5 Running Liberty

Liberty can be run from the UNIX System Services command line. However, if Liberty features require access to z/OS authorized services, this is supplied by a lightweight address space, fondly called the *Angel task* or *Angel process*. Only one Angel process is required per LPAR regardless of how many servers are running there.

As well as the Angel process, a server can also be a started task.

### 12.5.1 Using started tasks

Liberty supplies started task jobs for the Angel process and the server in the templates folder. See Figure 12-4 on page 180.



```
//BBGZSRV  PROC  PARMS='defaultServer'
 :
//*-------------------------------------------------------------------
//   SET INSTDIR='/usr/lpp/zWebSphere/V8R5/wlp'
//   SET USERDIR='/liberty_config'
//*-------------------------------------------------------------------
//STEP1    EXEC PGM=BPXBATSL,REGION=0M,
//   PARM='PGM &INSTDIR./lib/native/zos/s390x/bbgzsrv &PARMS'
//WLPUDIR   DD PATH='&USERDIR.'
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
 :
```

```
START BBGZSRV,PARMS='myServer'
```

*Figure 12-4 Liberty server started tasks*

### 12.5.2 Starting and stopping a Liberty Server as a started task

Before running the following commands, set up the “STARTED” and “SERVER” profiles:

- start BBGZSRV,JOBNAME=BBSERV1,PARMS=’myServer’
- stop BBSERV1

**Note:** Refer to the Liberty Profile Quick Start Guide: “z/OS definitions” section

## 12.6 Configuring Operational Decision Manager to run with Liberty

After a Liberty server is created, the next step is to replace the default Liberty configuration with one that configures Liberty to host Operational Decision Manager.

To create the customization data set for Liberty, use the WebSphere Application Server control statement. After the customization job is run, you have a set of members customized for your topology. The jobs to create the Liberty configuration for Operational Decision Manager are described in the next subsection.

### 12.6.1 Configuration jobs

Operational Decision Manager supplies three configuration jobs for Liberty:

- HBRWLPC
- HBRWLPR
- HBRWLPS

The `server.xml` file and the customized management file for the Rule Execution Server console is created by running the preceding jobs. More detail is given in the following sections.

Then, the .war files can be placed in the Liberty server's application directory and the `server.xml` file replaced with the one for Operational Decision Manager.

### 12.6.2 The server.xml file

The Liberty configuration file supplied by the HBRWLPC job after it is run, places a customized `server.xml` file in the working directory that is specified. Move this file to the directory of the Liberty server.

The Liberty features that are required by Operational Decision Manager are already defined in this configuration file.

### 12.6.3 Security profiles for Operational Decision Manager on Liberty

The RACF HBRWLPR sample job defines the following profiles:

- BBGZDFLT.res.resAdministrators
- BBGZDFLT.res.resDeployers
- BBGZDFLT.res.resMonitors
- BBGZDFLT.testing.resAdministrators
- BBGZDFLT.testing.resDeployers
- BBGZDFLT.DecisionRunner.resAdministrators
- BBGZDFLT.DecisionRunner.resDeployers

The first part of the name, BBGZDFLT, is the prefix name as given in the started task of Liberty.

The second part, for example, "res" is the ID of the application that is defined to Liberty. In this case, this is the Rule Execution Management Console. The final part of the name is the Operational Decision Manager group that users are allocated.

### 12.6.4 Connecting to the persistent store

Liberty can connect to a local database on the same LPAR using a Type 2 connection. This is the default setting for Operational Decision Manager. Using a Type 2 connection is the optimal way to connect to the database, but requires the Liberty Angel process to be running.

Run the Operational Decision Manager database jobs for defining the database to which the RES Management Console connects.

**Note:** The Liberty server can also be started from the UNIX System Services command line with this type 2 JDBC connection to the database.

### 12.6.5 Enabling the Operational Decision Manager applications in Liberty

Enable Liberty with the Operational Decision Manager function by copying the following application to the application directory of the Liberty server.

In the directory:

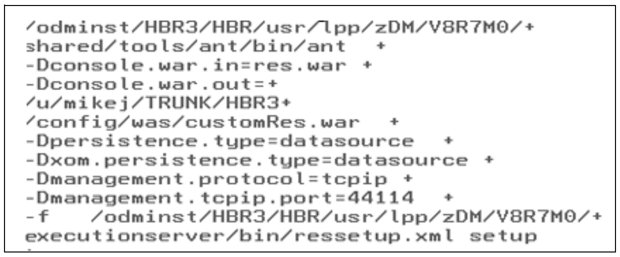
`/usr/lpp/zDM/V8R7M0/executionserver/applicationservers/WLP855`

- `testing.war`
- `DecisionRunner.war`
- `DecisionService.war`

Copy these three .war files to the application directory of the Liberty server.

The following .war file is input to the HBRWLPS job: `res.war`.

Figure 12-5 shows the input to the `ressetup.xml` Ant task. This job creates a Rule Execution Server console .war file updated with the specific values for this customization, and writes the output in the Operational Decision Manager working directory into the .war file: `customerRes.war`



```
/odminst/HBR3/HBR/usr/lpp/zDM/V8R7M0/+
shared/tools/ant/bin/ant   +
-Dconsole.war.in=res.war  +
-Dconsole.war.out=+
/u/mikej/TRUNK/HBR3+
/config/was/customRes.war   +
-Dpersistence.type=datasource   +
-Dxom.persistence.type=datasource  +
-Dmanagement.protocol=tcpip  +
-Dmanagement.tcpip.port=44114   +
-f   /odminst/HBR3/HBR/usr/lpp/zDM/V8R7M0/+
executionserver/bin/ressetup.xml setup
```

*Figure 12-5 RES console configuration with job* HBRWLPS

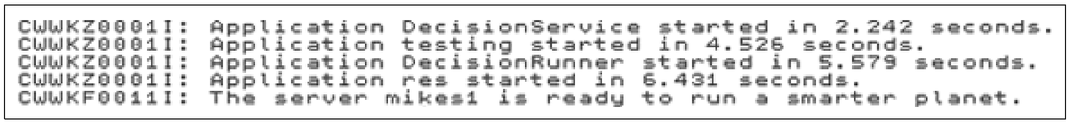
**Note:** Do not copy the `res.war` file to the Liberty server's application directory because this has not been customized to the local settings.

Copy the `customerRes.war` new file to the application directory of the Liberty server.

## Starting the Liberty server

Restart the Liberty server, which is now configured for Operational Decision Manager from either the started task or from the command line of UNIX System Services 12.5.2, "Starting and stopping a Liberty Server as a started task" on page 180.

If the server started as a started task, the following output is seen in the job log (Figure 12-6).



```
CWWKZ0001I: Application DecisionService started in 2.242 seconds.
CWWKZ0001I: Application testing started in 4.526 seconds.
CWWKZ0001I: Application DecisionRunner started in 5.579 seconds.
CWWKZ0001I: Application res started in 6.431 seconds.
CWWKF0011I: The server mikes1 is ready to run a smarter planet.
```

*Figure 12-6 Output in the servers started task job log*

The server is now running with Operational Decision Manager.

After the server comes up and the applications are started, use a web browser to connect to the Operational Decision Manager RES console using the example URL provided in the following shaded box.

**For example:** http://machinename.hursley.ibm.com:9111/res

13

# Configuring IBM WebSphere Optimized Local Adapters support

WebSphere Optimized Local Adapters (WOLA) is a feature of WebSphere Application Server for z/OS that manages communication between WebSphere Application Server and an external address space, such as CICS, batch, or IMS, that resides in the same logical partition (LPAR).

For more information about WOLA, see the *Configuring WebSphere Optimized Local Adapters (WOLA)* topic in the Operational Decision Manager Version 8.7.1 IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/con_ds_install_config_wola.html

The following topics are covered in this chapter:

## 13.1 Overview of WebSphere Operational Local Adapters

WebSphere Operational Local Adapters is a component of WebSphere Application Server for z/OS. It uses cross-memory mechanisms to provide a bidirectional, high volume exchange of messages between WebSphere Application Server for z/OS and the calling application.

When Operational Decision Manager is installed on WebSphere Application Server for z/OS, WOLA can be used as a way for COBOL or PL/I applications to execute rules by using Rule Execution Server (RES) within Operational Decision Manager on WebSphere Application Server for z/OS, rather than running the z/OS native Rule Execution Server. COBOL or PL/I applications do not need any changes to connect to Operational Decision Manager using WOLA because the redirection is achieved by runtime JCL variables. This allows Operational Decision Manager to benefit from this method of high volume message exchange.

### 13.1.1 Configuring WOLA

When calling to RES using WOLA, it is necessary for WOLA to know the correct WebSphere Application Server with which to connect. There might be more than one WebSphere Application Server running on the same system and you need to connect to the one running RES. This information is included in the required JCL variables.

### 13.1.2 JCL variables for using WOLA

Use the following JCL variables for WOLA:

- The first variable indicates that the target RES is accessed using WOLA. Set *++HBRTARGETRES++* to `WOLA`.
- RES also needs to know where the WOLA load library can be located. This is the load library that was created as part of the WOLA setup. The *++HBRWOLALOADLIB++* indicates the location of your WOLA load library. This load library is created as part of the WOLA configuration process.
- Indicate the details of the WebSphere Application Server to use by using the Cell, Node, and Server name, which lead to a unique WebSphere Application Server. The following variables give a unique identifier to the correct WebSphere Application Server:
  - *++HBRWOLACELL++*
  - *++HBRWOLANODE++*
  - *++HBRWOLASERVER++*

For more information about the WOLA-related variables that can be set in the HBRINST member, see “WebSphere Optimized Local Adapters script parameters” on page 237.

## 13.2 Configuration of WebSphere Application Server to use WOLA

The exact configuration of WOLA depends on the version of WebSphere Application Server for z/OS that is used.

For more information, see the *Configuring WebSphere Optimized Local Adapters (WOLA)* topic in the Operational Decision Manager Version 8.7 IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/tsk_ds_install_wola_was8.html

Assume that WOLA is configured for WebSphere Application Server with the load libraries created at `WODM.OLA.LOADLIB`.

This section provides an example of a configuration that only uses the following steps to configure WOLA:

1. Create a load library that contains the modules that are required by WOLA from within the WebSphere Application Server installation directory in UNIX System Services. See the WebSphere Application Server documentation for more details. In this example, it is assumed that the load libraries have been created at `WODM.OLA.LOADLIB`.
2. Install and configure WOLA as described in the product documentation for your version of WebSphere Application Server.
3. Restart WebSphere Application Server to pick up the changes. You see messages in the WebSphere Application Server logs that indicate the WOLA status:

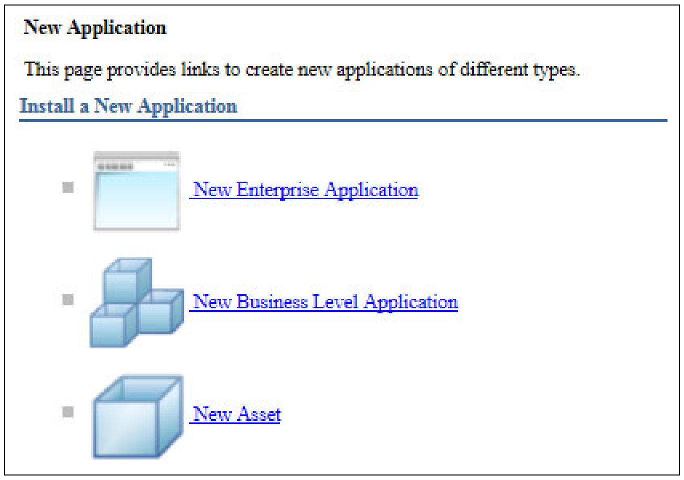
   ```
   Support is activated: BBOM0001I enable_adapter:1
   ```
4. It is now necessary to install the WOLA Enterprise JavaBeans (EJB) into WebSphere Application Server. This application is used to listen for the WOLA input.

   **Note:** The following instructions are specific for WebSphere Application Server V8.5. If you are using a different version, other configuration might be necessary. For more information, see the IBM Knowledge Center:

   http://www-01.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/tsk_ds_deploy_wola_ear_was.html

   Complete the following steps:

   a. In the WebSphere Application Server administrative console, select **Application** → **New Application** and select **New Enterprise Application** from the New Application panel, as shown in Figure 13-1.


New Application

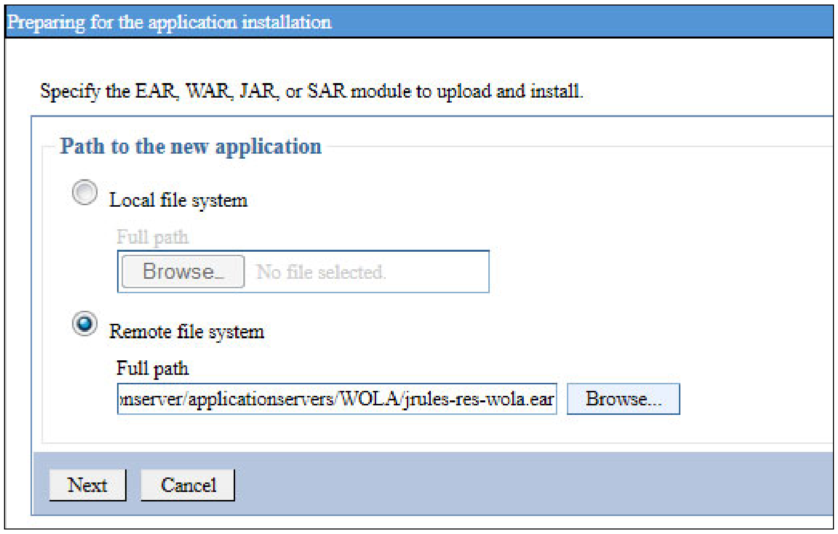
This page provides links to create new applications of different types.

Install a New Application

- New Enterprise Application
- New Business Level Application
- New Asset

*Figure 13-1 WebSphere Application Server New Application panel*

b. On the “Preparing for the application installation” panel, select **Remote file system** and click **Browse** to help you select the location of the WOLA EAR file on the z/OS server. It is in the `/executionserver/applicationservers/WOLA` directory of your installation. An example is shown in Figure 13-2.



Preparing for the application installation

Specify the EAR, WAR, JAR, or SAR module to upload and install.

Path to the new application

Local file system

Full path

Browse... No file selected.

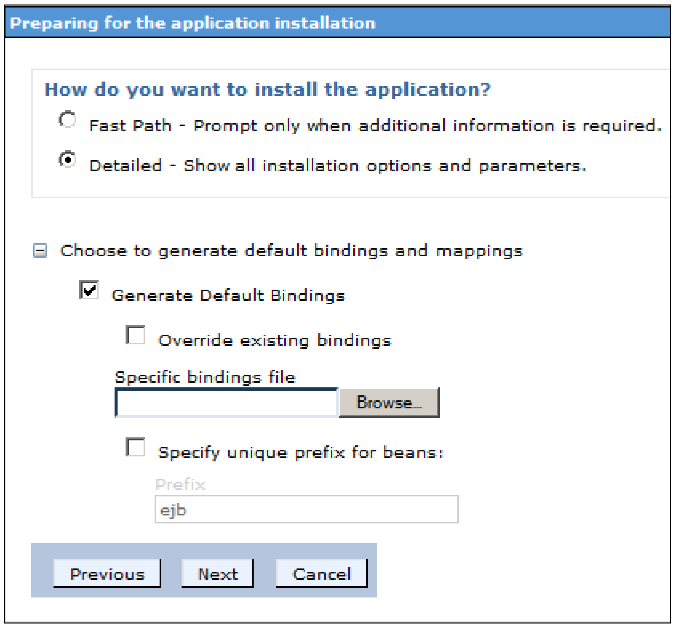
[✓ Remote file system]

Full path

nserver/applicationservers/WOLA/jrules-res-wola.ear

*Figure 13-2 Preparing for the application installation*

c. On the following window, select **Detailed** for the type of installation. Expand **Choose to generate default bindings and mappings** and check **Generate Default Bindings**, as shown in Figure 13-3. Click **Next**.



Preparing for the application installation

How do you want to install the application?

Fast Path - Prompt only when additional information is required.

[✓ Detailed - Show all installation options and parameters.]

Choose to generate default bindings and mappings

[✓ Generate Default Bindings]

Override existing bindings

Specific bindings file

Specify unique prefix for beans:

Prefix

ejb

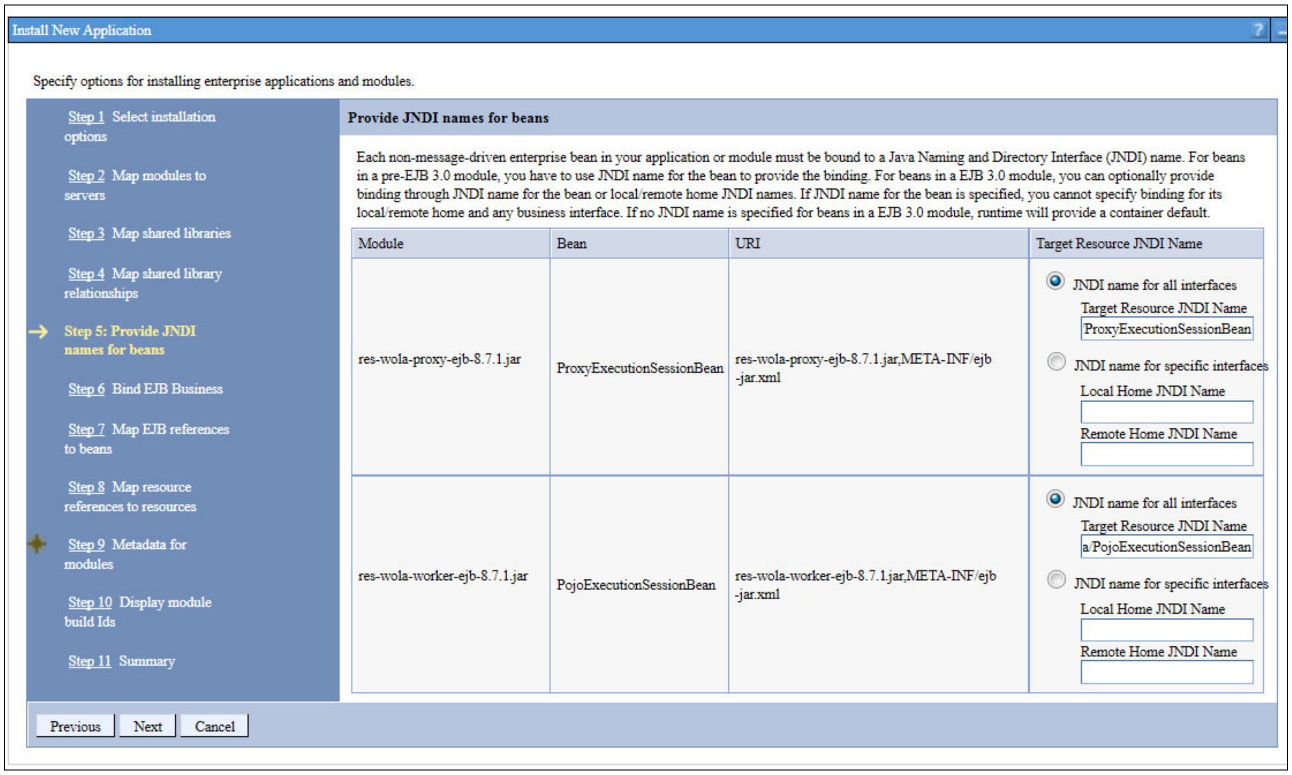
*Figure 13-3 Selecting the options for installing the application*

d. On the Install New Application window, click **Next** until you reach step 5.

e. For `res-wola-proxy-ejb-8.7.1.jar`, select **JNDI for all interfaces**. If it is not already entered, enter `ejb/com/ibm/rules/wola/ProxyExecutionSessionBean` for the Target Resource JNDI Name.

For `res-wola-worker-ejb-8.7.1.jar`, select **JNDI name for all interfaces**. If it is not already entered, enter `ejb/com/ibm/rules/wola/PojoExecutionSessionBean` for the Target Resource JNDI Name.

Figure 13-4 shows the step to provide JNDI names for beans. Then, click **Next** until you reach step 7.


Install New Application
Step 1 Select installation options
Step 2 Map modules to servers
Step 3 Map shared libraries
Step 4 Map shared library relationships
Step 5: Provide JNDI names for beans
Step 6 Bind EJB Business
Step 7 Map EJB references to beans
Step 8 Map resource references to resources
Step 9 Metadata for modules
Step 10 Display module build Ids
Step 11 Summary
Previous
Next
Cancel

Specify options for installing enterprise applications and modules.

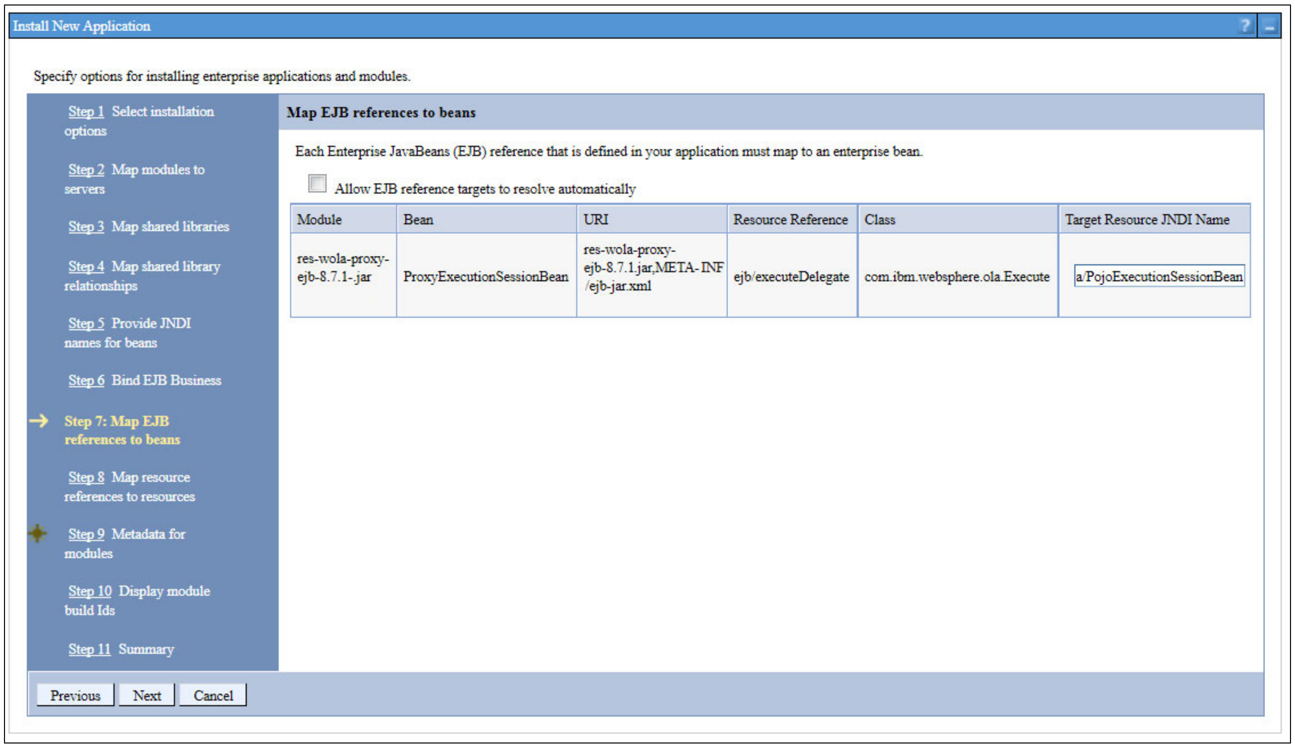
**Provide JNDI names for beans**

Each non-message-driven enterprise bean in your application or module must be bound to a Java Naming and Directory Interface (JNDI) name. For beans in a pre-EJB 3.0 module, you have to use JNDI name for the bean to provide the binding. For beans in a EJB 3.0 module, you can optionally provide binding through JNDI name for the bean or local/remote home JNDI names. If JNDI name for the bean is specified, you cannot specify binding for its local/remote home and any business interface. If no JNDI name is specified for beans in a EJB 3.0 module, runtime will provide a container default.

| Module | Bean | URI | Target Resource JNDI Name |
|---|---|---|---|
| res-wola-proxy-ejb-8.7.1.jar | ProxyExecutionSessionBean | res-wola-proxy-ejb-8.7.1.jar,META-INF/ejb-jar.xml | [● JNDI name for all interfaces] Target Resource JNDI Name: ProxyExecutionSessionBean; ○ JNDI name for specific interfaces; Local Home JNDI Name: ; Remote Home JNDI Name: |
| res-wola-worker-ejb-8.7.1.jar | PojoExecutionSessionBean | res-wola-worker-ejb-8.7.1.jar,META-INF/ejb-jar.xml | [● JNDI name for all interfaces] Target Resource JNDI Name: a/PojoExecutionSessionBean; ○ JNDI name for specific interfaces; Local Home JNDI Name: ; Remote Home JNDI Name: |

*Figure 13-4 Step 5: Provide JNDI names for beans*

f. Step 7 requires the mapping of EJB references to beans. For the Module column for `res-wola-proxy-ejb-8.7.1.jar`, if it is not already entered, enter the Target Resource JNDI Name of `ejb/com/ibm/rules/wola/PojoExecutionSessionBean`, as shown in Figure 13-5. Click **Next** until you reach the Summary window.


Install New Application
Step 1 Select installation options
Step 2 Map modules to servers
Step 3 Map shared libraries
Step 4 Map shared library relationships
Step 5 Provide JNDI names for beans
Step 6 Bind EJB Business
Step 7: Map EJB references to beans
Step 8 Map resource references to resources
Step 9 Metadata for modules
Step 10 Display module build Ids
Step 11 Summary
Previous
Next
Cancel

Specify options for installing enterprise applications and modules.

**Map EJB references to beans**

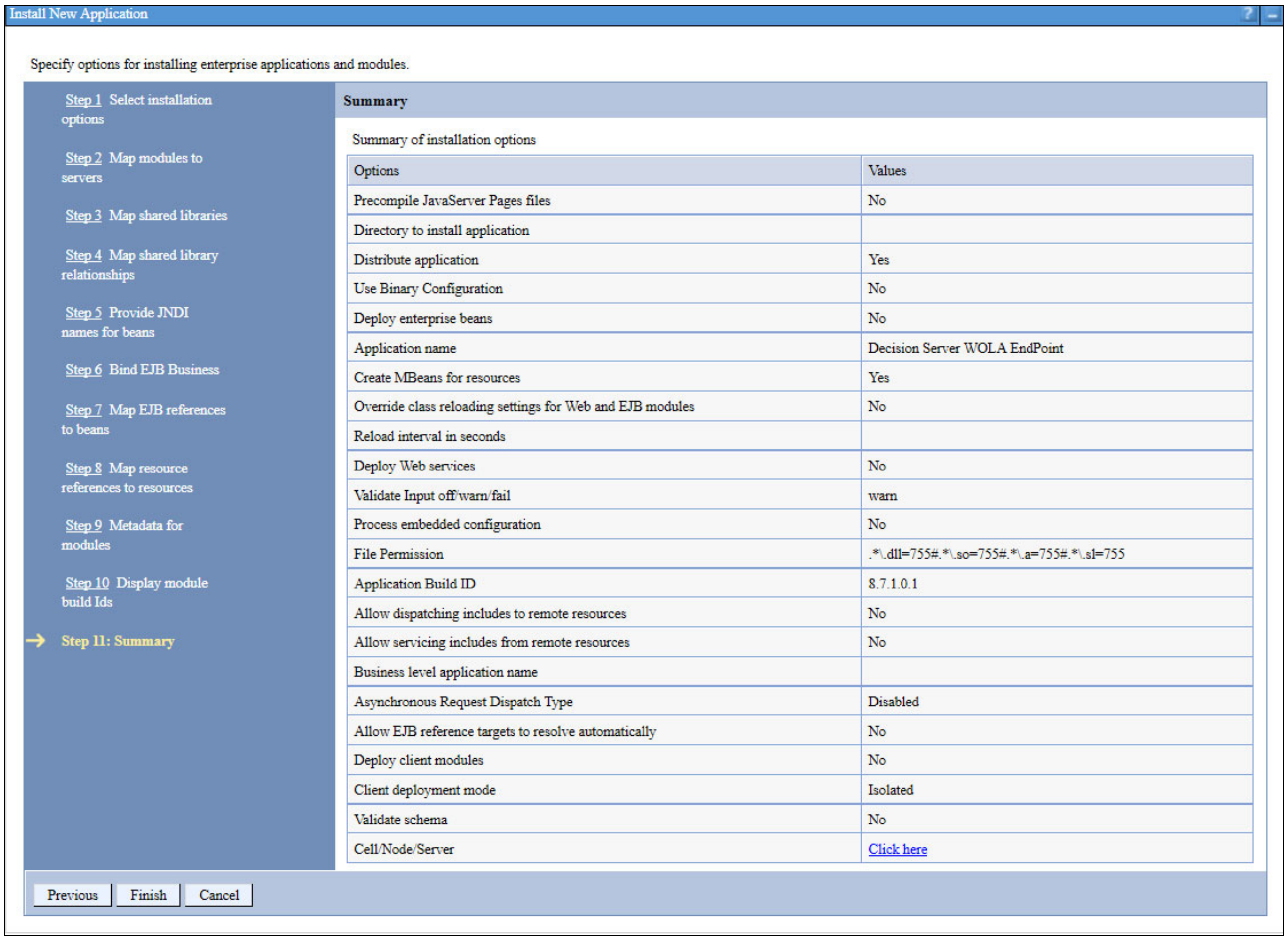
Each Enterprise JavaBeans (EJB) reference that is defined in your application must map to an enterprise bean.

☐ Allow EJB reference targets to resolve automatically

| Module | Bean | URI | Resource Reference | Class | Target Resource JNDI Name |
|---|---|---|---|---|---|
| res-wola-proxy-ejb-8.7.1-.jar | ProxyExecutionSessionBean | res-wola-proxy-ejb-8.7.1.jar,META-INF/ejb-jar.xml | ejb/executeDelegate | com.ibm.websphere.ola.Execute | a/PojoExecutionSessionBean |

*Figure 13-5 Step 7: Mapping the EJB reference to the bean*

g. On the Summary window, as shown in Figure 13-6, click **Finish**. The application is installed. Click **Save** to save the changes.


Install New Application
Step 1 Select installation options
Step 2 Map modules to servers
Step 3 Map shared libraries
Step 4 Map shared library relationships
Step 5 Provide JNDI names for beans
Step 6 Bind EJB Business
Step 7 Map EJB references to beans
Step 8 Map resource references to resources
Step 9 Metadata for modules
Step 10 Display module build Ids
Step 11: Summary
Previous
Finish
Cancel

Specify options for installing enterprise applications and modules.

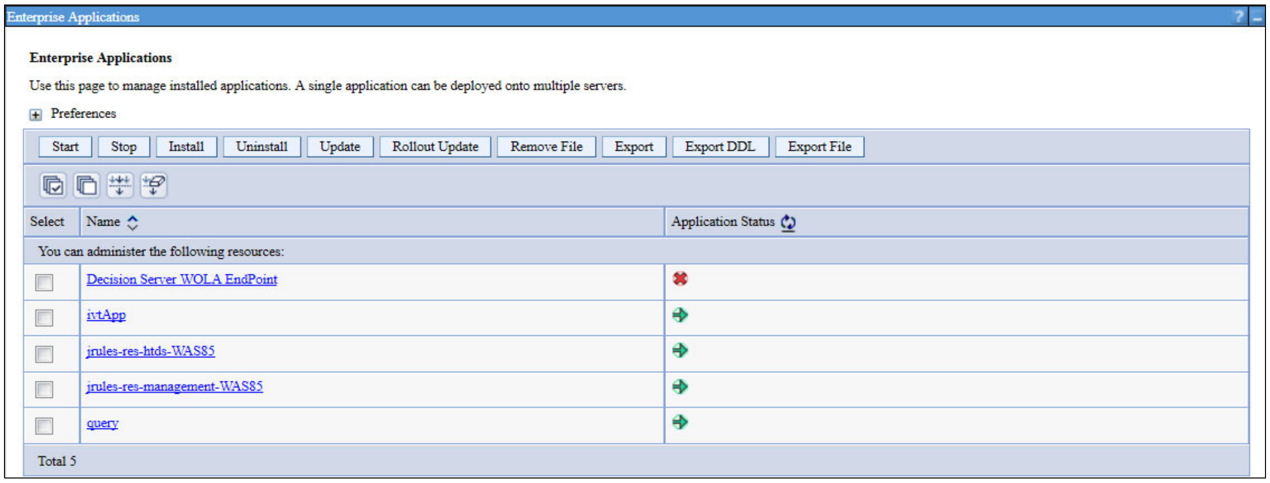
Summary

Summary of installation options

| Options | Values |
|---|---|
| Precompile JavaServer Pages files | No |
| Directory to install application | |
| Distribute application | Yes |
| Use Binary Configuration | No |
| Deploy enterprise beans | No |
| Application name | Decision Server WOLA EndPoint |
| Create MBeans for resources | Yes |
| Override class reloading settings for Web and EJB modules | No |
| Reload interval in seconds | |
| Deploy Web services | No |
| Validate Input off/warn/fail | warn |
| Process embedded configuration | No |
| File Permission | .*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755 |
| Application Build ID | 8.7.1.0.1 |
| Allow dispatching includes to remote resources | No |
| Allow servicing includes from remote resources | No |
| Business level application name | |
| Asynchronous Request Dispatch Type | Disabled |
| Allow EJB reference targets to resolve automatically | No |
| Deploy client modules | No |
| Client deployment mode | Isolated |
| Validate schema | No |
| Cell/Node/Server | Click here |

*Figure 13-6 Step 11: Summary of application installation*

h. The application can be displayed by selecting **Applications** → **Application Types** → **WebSphere Enterprise Applications**, as shown in Figure 13-7, where the application is listed as Decision Server WOLA EndPoint. Start the application by selecting it and clicking **Start**.



Use this page to manage installed applications. A single application can be deployed onto multiple servers.

| Select | Name | Application Status |
|---|---|---|
| You can administer the following resources: | | |
| ☐ | Decision Server WOLA EndPoint | ✖ |
| ☐ | ivtApp | ➔ |
| ☐ | jrules-res-htds-WAS85 | ➔ |
| ☐ | jrules-res-management-WAS85 | ➔ |
| ☐ | query | ➔ |
| Total 5 | | |

*Figure 13-7 List of installed applications that include WOLA EndPoint*

## 13.3 Batch programs and Rule Execution Server using WOLA on z/OS

With the previous configurations already in place (setting up the variables as described in 13.1, “Overview of WebSphere Operational Local Adapters” on page 184, and setting up the WebSphere Application Server as described in 13.2, “Configuration of WebSphere Application Server to use WOLA” on page 184), there are no further requirements to connect a batch program to RES running in WebSphere Application Server for z/OS using WOLA.

## 13.4 CICS and Rule Execution Server using WOLA on z/OS

Several of the steps described in this section are the same steps that are required for setting up CICS to work with RES. These steps are indicated. However, there are some differences to ensure that CICS is aware of the location of WOLA and the details about which WebSphere Application Server for z/OS that WOLA is accessing. Complete the following steps:

1. Check the value of the **`ola_cicsuser_identity_propagate`** variable. This variable is used to specify permissions for CICS application level identities to be used for authentication when calling the rule. By default, it is set to `0`, which indicates undefined. More information about this variable is in the *Optimized local adapters environment variables* topic of the WebSphere Application Server, Network Deployment, Version 8.5 IBM Knowledge Center:

   `http://www-01.ibm.com/support/knowledgecenter/SSEQTJ_8.5.5/com.ibm.websphere.zseries.doc/ae/cdat_olacustprop.html`

2. Similar to setting up CICS for RES, submit the JCL job HBRCSD to define the resources that are required by CICS.

3. Submit the HBRCWOLA job. This job defines the resources that are required for WOLA under CICS.

4. Edit the CICS system initialization table. For setting up CICS for RES, add the value of the **`CICSLIST`** parameter to the list of resource definition groups specified by the **`GRPLIST`** parameter. To work with WOLA, also add the name BBOLIST to the **`GRPLIST`** parameter to ensure that CICS can locate the necessary transactions.
5. When setting up CICS for RES, the `SHBRCICS` load libraries needed to be added to DFHRPL concatenation in the CICS JCL. To work with WOLA, you also need to add the `SHBRWOLA` libraries, for example:

```
//DFHRPL DD DISP=SHR,DSN=&HBRHLQ..SHBRCICS
// DD DISP=SHR,DSN=WODM.OLA.LOADLIB
```

6. Pass the necessary runtime variables to the server by adding the SHBRPARM(HBRWOLA) and SHBRPARM(HBRCMMN) data set members to the HBRENVPR data definition (DD) statement, as shown in the following DD statements:

```
//HBRENVPR DD DISP=SHR,DSN=++HBRHLQ++.SHBRPARM(HBRWOLA)
//         DD DISP=SHR,DSN=++HBRHLQ++.SHBRPARM(HBRCMMN)
```

7. If you plan to run the Miniloan sample application to verify your configuration, add the SHBRPARM(HBRSCEN) data set member as a SCENARIO DD statement. The HBRSCEN member contains input values to the Miniloan application.

```
//SCENARIO DD DISP=SHR,DSN=++HBRWORKDS++.SHBRPARM(HBRSCEN)
```

8. Start the WebSphere Application Server and CICS.
9. RES requires the HBRGROUP resources to be installed. WOLA requires the BBOACSD group to be installed. Use the following commands:

```
CEDA INSTALL GROUP(HBRGROUP)
CEDA INSTALL GROUP(BBOACSD)
```

10. Activate the optimized local adapters TRUE program by using the following command:

```
BBOC START_TRUE
```

11. The configuration can be tested by running the HBRC transaction from CICS.

For more information, see the *Use the CICS environment* topic in the WebSphere Application Server, Network Deployment, Version 8.5 IBM Knowledge Center:

http://www-01.ibm.com/support/knowledgecenter/SS7K4U_8.5.5/com.ibm.websphere.zseries.doc/ae/tdat_useola_in_step4.html

## 13.5 IMS and Rule Execution Server using WOLA on z/OS

To connect IMS to RES using a WOLA interface, it is necessary to provide IMS with details of the location of that WOLA. In each case, follow this process:

1. There needs to be an entry in the external subsystem member in the IMS PROCLIB to contain an entry to indicate that WOLA must be used. If you do not already have a member, you need to create one. Include this entry:

```
WOLA,BBOA,BBOAIEMT
```

2. Pass the **`SSM`** parameter into your IMS start data.

3. The WOLA load library that is created during the WOLA setup needs to be included in the IMS control region startup JCL in both the `STEPLIB` and the `DFSESL` DDs:
   - BMP: You need to restart IMS in order to pick up the changes.
   - MPR: MPR requires additional steps for setup:
     i. Similar to setting up for calling out to RES (as described in 11.2, "Configuration" on page 170), include an `HBRENVPR DD` in your message processing region JCL. In this case, ensure that it points to a member that contains the WOLA parameters rather than the RES group.
     ii. Similar to setting up for calling out to RES, add the WOLA load library to your `STEPLIBR` and `DFSESL DD` statements in the message processing JCL.

   The message processing region needs to be restarted for IMS to pick up the changes:
   - DL/I: DL/I programs are currently not supported in this environment.

For information, see the *Use the IMS environment* topic in the WebSphere Application Server, Network Deployment, Version 8.5 IBM Knowledge Center:

`http://www-01.ibm.com/support/knowledgecenter/SS7K4U_8.5.5/com.ibm.websphere.zseries.doc/ae/tdat_useola_in_step5.html`

14

# Configuring Decision Warehousing

This chapter describes the Decision Warehouse, which is part of the Rule Execution Server. The configuration of the Decision Warehouse and its possible uses are also covered.

The following topics are covered in this chapter:

## 14.1 Introducing the Decision Warehouse

The Decision Warehouse is used for monitoring and reporting on ruleset execution. With ruleset monitoring enabled, details of each ruleflow used, the path, and the rules fired are recorded. The purpose of the Decision Warehouse function is to help you understand what happened when a ruleset was executed. This data might be required for auditing or performance analysis.

The Decision Warehouse is accessed by using the Rule Execution Server (RES) console[1]. The details of recorded statistics are stored in a database for future reporting.

Decision Warehouse is shipped in a default configuration so that it can be used immediately.

## 14.2 Configuring the Decision Warehouse

Complete the following steps to configure the Decision Warehouse:

- Set up the database resources
- Enable ruleset monitoring

### 14.2.1 Setting up the database resources

During the normal Operational Decision Manager configuration, the Decision Warehouse database definitions, customized to your environment for the Decision Warehouse function, are presented in the partitioned data set (PDS) member:

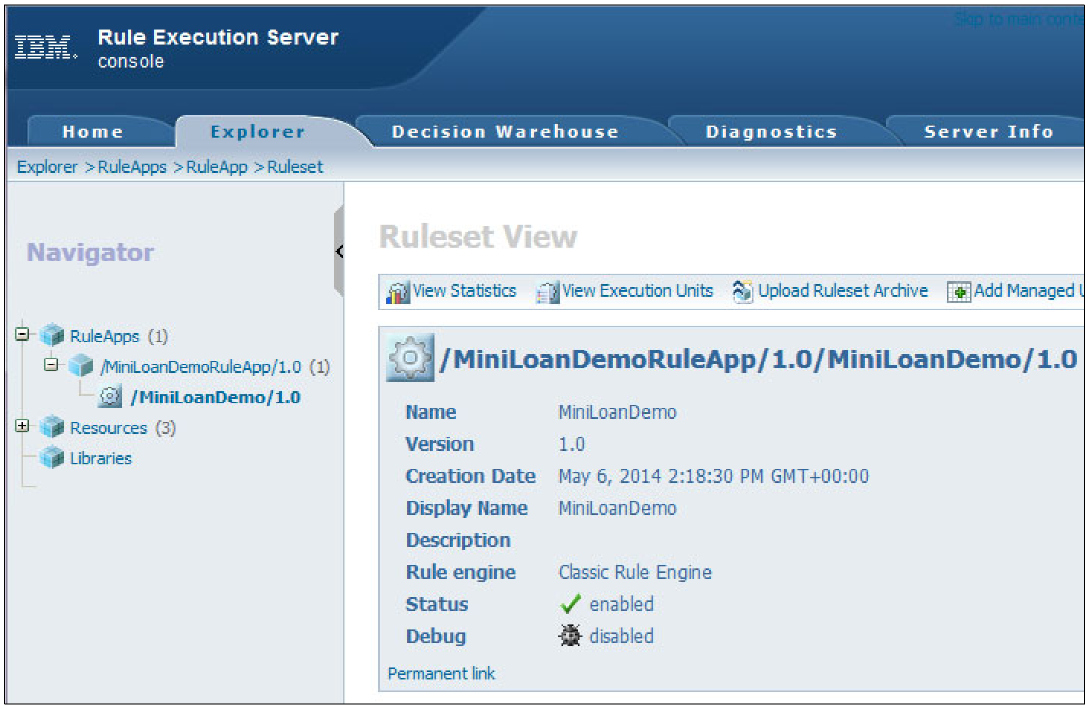
```
++HBRWORKDS++.SHBRJCL(HBRDSCTR)
```

Run the HBRDSCTR job to create the database entities that are required to store the rules data that is traced during rule execution.

### 14.2.2 Enabling ruleset monitoring

To monitor ruleset execution, you must set the monitoring options in the ruleset parameters, available from the Rule Execution Server console, a section of which is shown in Figure 14-1 on page 195.

[1] Recall that the console is the same irrespective of whether it is being hosted via zRule Execution Server for z/OS (zRES) or via the console for Rule Execution Server (RES) on WebSphere Application Server for z/OS.



/MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0

| | |
|---|---|
| Name | MiniLoanDemo |
| Version | 1.0 |
| Creation Date | May 6, 2014 2:18:30 PM GMT+00:00 |
| Display Name | MiniLoanDemo |
| Description | |
| Rule engine | Classic Rule Engine |
| Status | ✓ enabled |
| Debug | disabled |

*Figure 14-1 Ruleset view of the Rule Execution Console*

To set up monitoring, follow these steps in the RES console, note Figure 14-1:

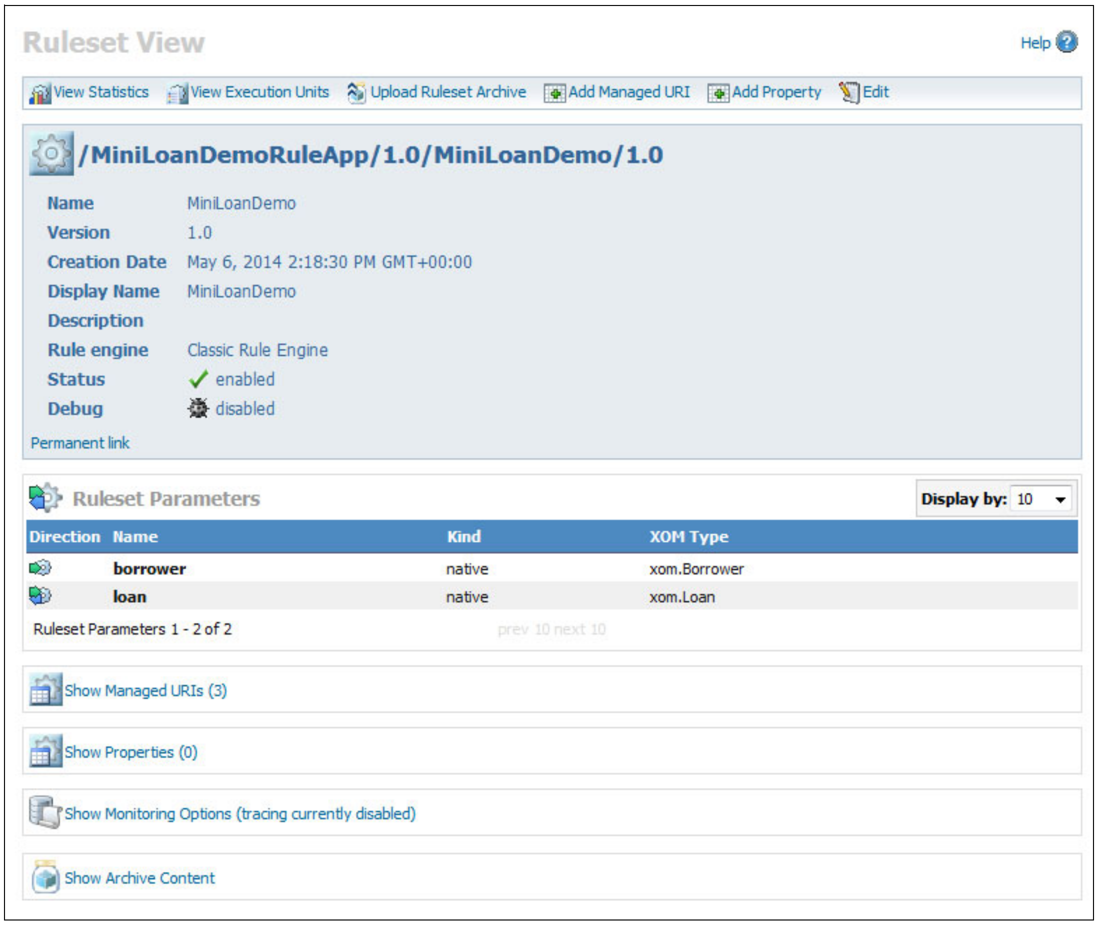
1. On the Rule Execution Server console window, select the **Explorer** tab.

From the **Navigator** pane:

2. Expand **RuleApps** to see a list of RuleApps in the **RuleApp** view.
3. Expand a RuleApp to see the rulesets.

4. Select a ruleset to see the **Ruleset View** as shown in Figure 14-2.

**Note:** The boxed area **Show Monitoring Options (tracing currently disabled)** is collapsed in the diagram.



/MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0

| | |
|---|---|
| Name | MiniLoanDemo |
| Version | 1.0 |
| Creation Date | May 6, 2014 2:18:30 PM GMT+00:00 |
| Display Name | MiniLoanDemo |
| Description | |
| Rule engine | Classic Rule Engine |
| Status | ✓ enabled |
| Debug | disabled |

Permanent link

Ruleset Parameters

Display by: 10

| Direction | Name | Kind | XOM Type |
|---|---|---|---|
| | borrower | native | xom.Borrower |
| | loan | native | xom.Loan |

Ruleset Parameters 1 - 2 of 2 prev 10 next 10

Show Managed URIs (3)

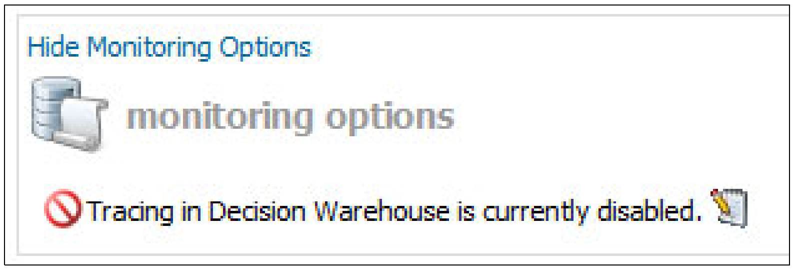
Show Properties (0)

Show Monitoring Options (tracing currently disabled)

Show Archive Content

Figure 14-2 Ruleset View

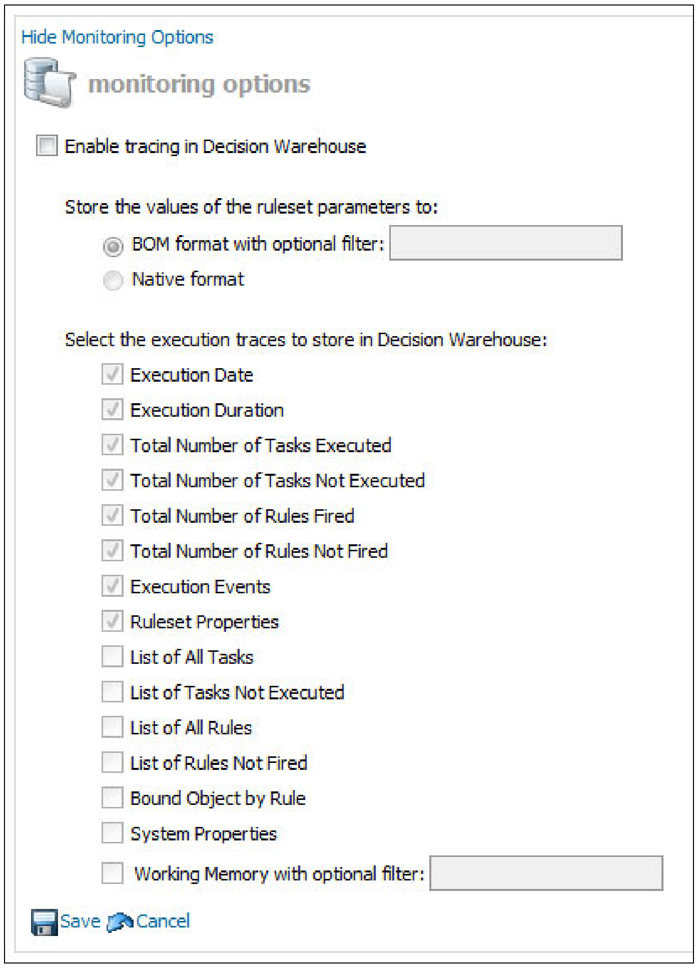
5. Click **Show Monitoring Options (tracing currently disabled)** to expand the box and present the box as shown in Figure 14-3.



Tracing in Decision Warehouse is currently disabled.

Figure 14-3 Hide Monitoring Options

6. Click the edit icon ( ) to present the monitoring options showing all the monitoring options that are available, as shown in Figure 14-4.



Hide Monitoring Options

monitoring options

[ ] Enable tracing in Decision Warehouse

Store the values of the ruleset parameters to:

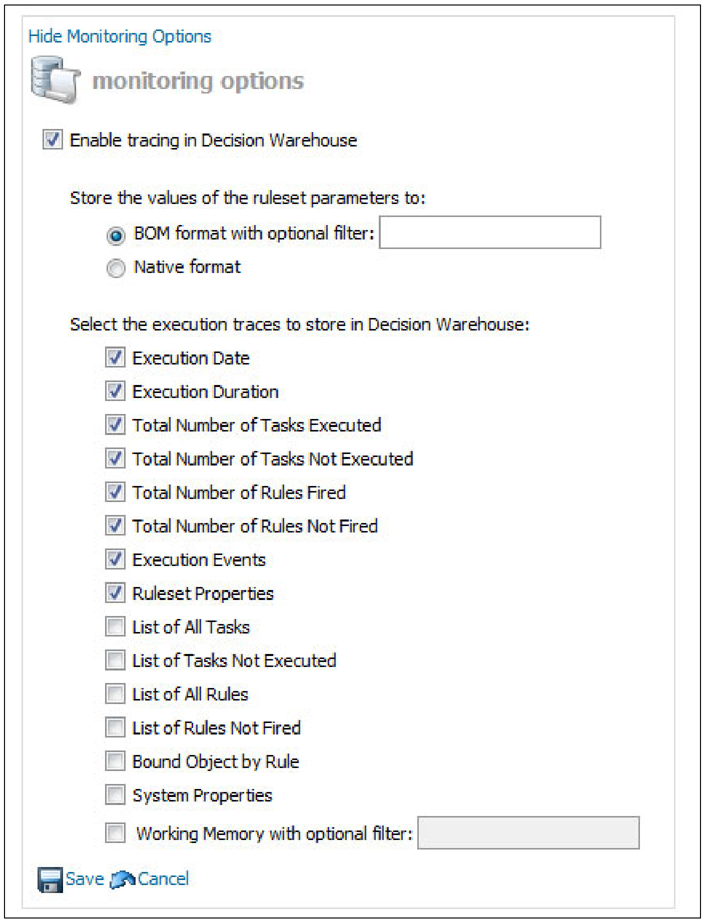
[● BOM format with optional filter:]
○ Native format

Select the execution traces to store in Decision Warehouse:

- [✓] Execution Date
- [✓] Execution Duration
- [✓] Total Number of Tasks Executed
- [✓] Total Number of Tasks Not Executed
- [✓] Total Number of Rules Fired
- [✓] Total Number of Rules Not Fired
- [✓] Execution Events
- [✓] Ruleset Properties
- [ ] List of All Tasks
- [ ] List of Tasks Not Executed
- [ ] List of All Rules
- [ ] List of Rules Not Fired
- [ ] Bound Object by Rule
- [ ] System Properties
- [ ] Working Memory with optional filter:

*Figure 14-4 Monitoring options*

7. To turn on monitoring and to enable specific options to be chosen, select **Enable tracing in Decision Warehouse**, resulting in Figure 14-5.


Hide Monitoring Options

monitoring options

- [✓] Enable tracing in Decision Warehouse

Store the values of the ruleset parameters to:

- ● BOM format with optional filter:
- ○ Native format

Select the execution traces to store in Decision Warehouse:

- [✓] Execution Date
- [✓] Execution Duration
- [✓] Total Number of Tasks Executed
- [✓] Total Number of Tasks Not Executed
- [✓] Total Number of Rules Fired
- [✓] Total Number of Rules Not Fired
- [✓] Execution Events
- [✓] Ruleset Properties
- [ ] List of All Tasks
- [ ] List of Tasks Not Executed
- [ ] List of All Rules
- [ ] List of Rules Not Fired
- [ ] Bound Object by Rule
- [ ] System Properties
- [ ] Working Memory with optional filter:

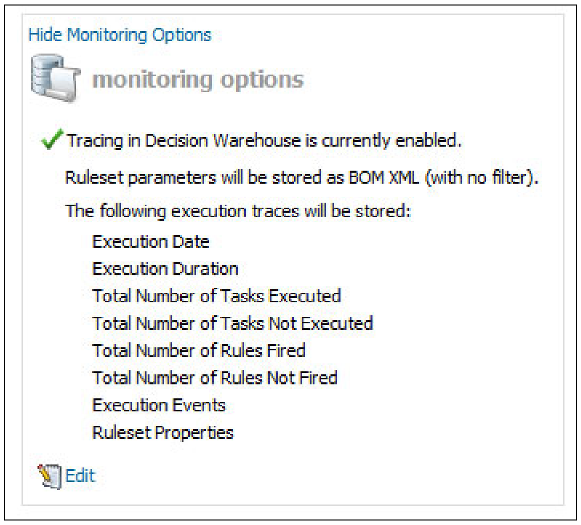
*Figure 14-5 Editable monitoring options*

8. Choose the monitoring options that you prefer.

**Note:** For more information about the Decision Warehouse monitoring options, see the *Decision Warehouse* section of IBM Knowledge Center for Operational Decision Manager 8.7:

`http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.dserver.rules.res.managing/topics/con_res_dw_overview.html`

9. When you have all the values set as required, save the settings by using the Save icon, resulting in a view that is similar to Figure 14-6.



✓ Tracing in Decision Warehouse is currently enabled.

Ruleset parameters will be stored as BOM XML (with no filter).

The following execution traces will be stored:

Execution Date
Execution Duration
Total Number of Tasks Executed
Total Number of Tasks Not Executed
Total Number of Rules Fired
Total Number of Rules Not Fired
Execution Events
Ruleset Properties

*Figure 14-6 Monitoring options set*

## 14.3 Viewing the results of running with Decision Warehousing enabled

To see the results of running rules with Decision Warehouse enabled, first appropriate rules need to be run, then the results can be read:

- 14.3.1, "Execute the appropriate ruleset" on page 199
- 14.3.2, "Viewing the results" on page 200

### 14.3.1 Execute the appropriate ruleset

Submit a job that calls the Operational Decision Manager runtime engine with rules that you chose to monitor.

**Note:** The execution trace data is now written to the default Decision Warehouse database until such times as tracing is disabled.

## 14.3.2 Viewing the results

Results of the tracing can be searched from the RES console.

1. From the RES console, select the **Decision Warehouse** tab (Figure 14-7).

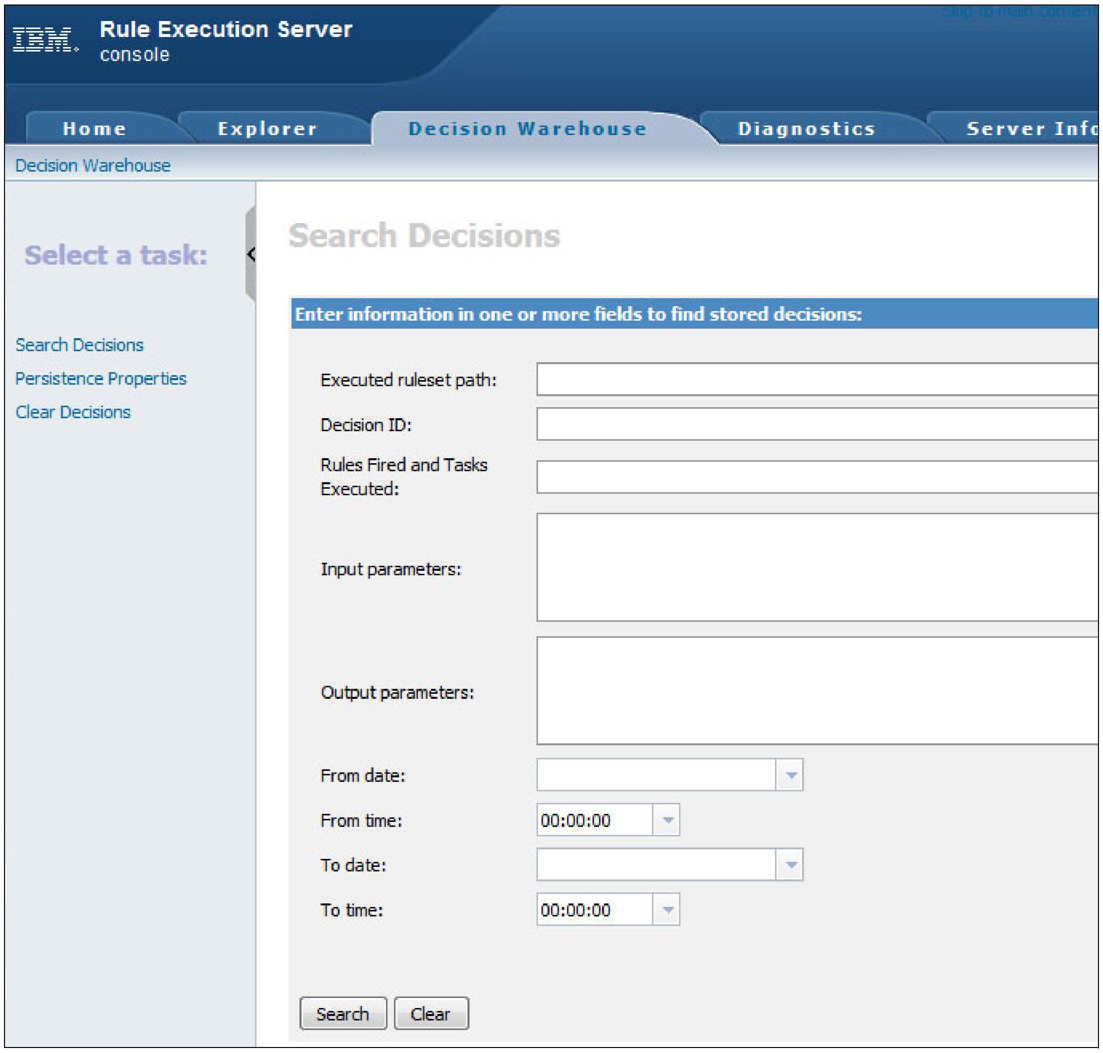
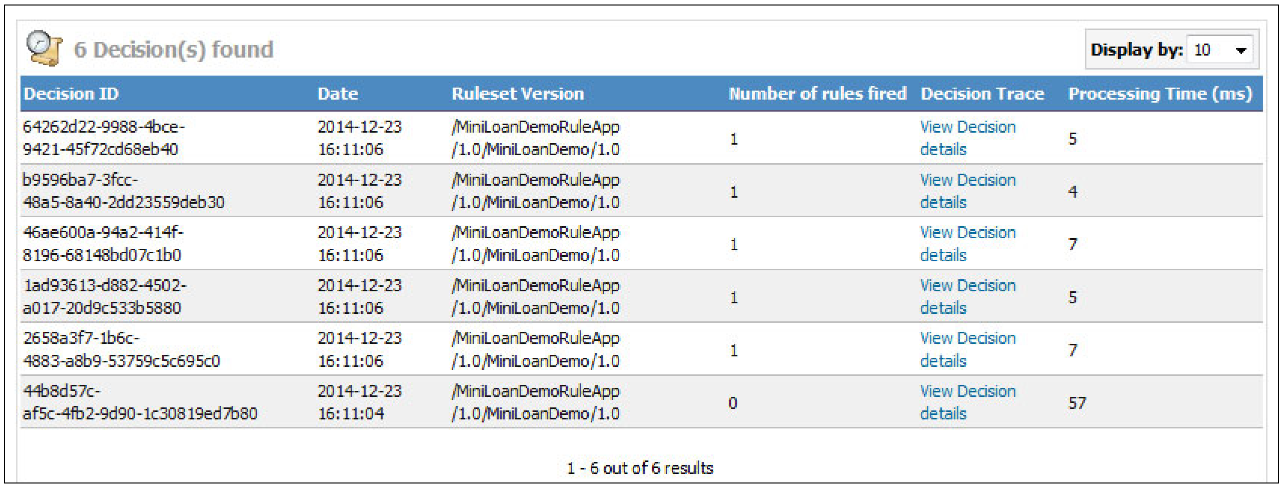


*Figure 14-7 Search Decisions view in the Rule Execution Server console*

2. Clicking **Search** provides a list of results, for example, see Figure 14-8.

**Note:** A decision identifier is automatically generated, by default, and is equal to the identifier of the execution unit (XU) connection.



6 Decision(s) found

Display by: 10

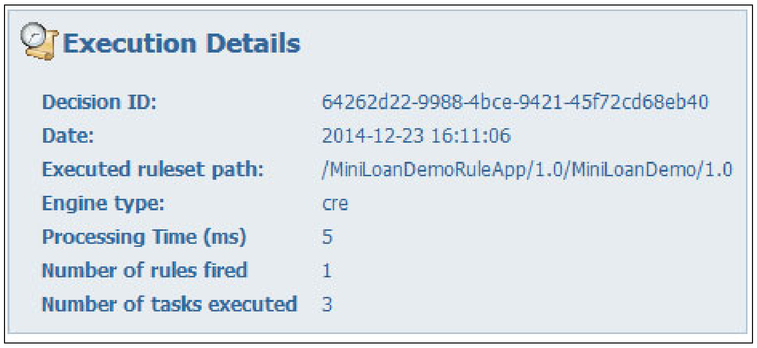
| Decision ID | Date | Ruleset Version | Number of rules fired | Decision Trace | Processing Time (ms) |
|---|---|---|---|---|---|
| 64262d22-9988-4bce-9421-45f72cd68eb40 | 2014-12-23 16:11:06 | /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0 | 1 | View Decision details | 5 |
| b9596ba7-3fcc-48a5-8a40-2dd23559deb30 | 2014-12-23 16:11:06 | /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0 | 1 | View Decision details | 4 |
| 46ae600a-94a2-414f-8196-68148bd07c1b0 | 2014-12-23 16:11:06 | /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0 | 1 | View Decision details | 7 |
| 1ad93613-d882-4502-a017-20d9c533b5880 | 2014-12-23 16:11:06 | /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0 | 1 | View Decision details | 5 |
| 2658a3f7-1b6c-4883-a8b9-53759c5c695c0 | 2014-12-23 16:11:06 | /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0 | 1 | View Decision details | 7 |
| 44b8d57c-af5c-4fb2-9d90-1c30819ed7b80 | 2014-12-23 16:11:04 | /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0 | 0 | View Decision details | 57 |

1 - 6 out of 6 results

*Figure 14-8 Example results list*

3. Click a link in the **Decision Trace** column to see a separate web page with full details about the rule execution in the following sections:

   a. Execution Details (Figure 14-9)

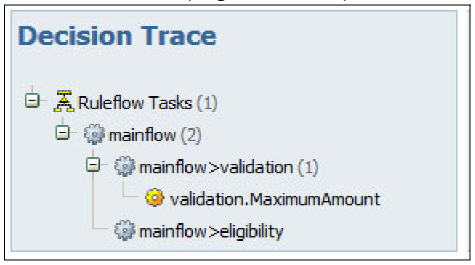


Execution Details

| | |
|---|---|
| Decision ID: | 64262d22-9988-4bce-9421-45f72cd68eb40 |
| Date: | 2014-12-23 16:11:06 |
| Executed ruleset path: | /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0 |
| Engine type: | cre |
| Processing Time (ms) | 5 |
| Number of rules fired | 1 |
| Number of tasks executed | 3 |

*Figure 14-9 Decision Trace Execution Details*

   b. Decision Trace (Figure 14-10)

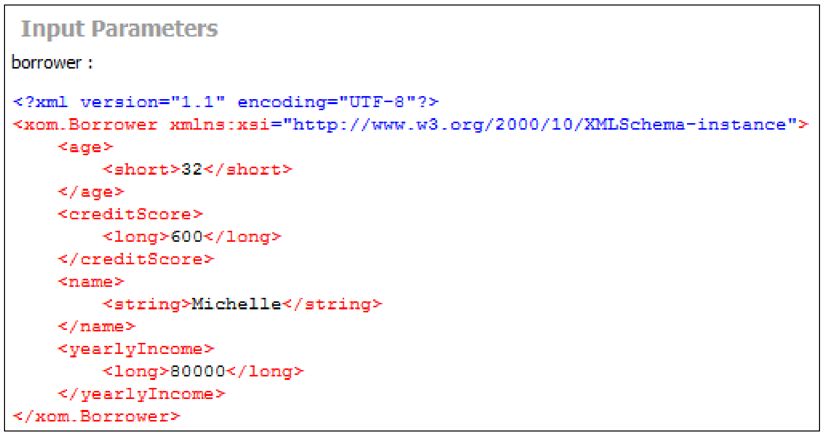


Decision Trace

```
⊟ Ruleflow Tasks (1)
   ⊟ mainflow (2)
      ⊟ mainflow>validation (1)
            validation.MaximumAmount
         mainflow>eligibility
```

*Figure 14-10 Decision Trace details*

c. Input Parameters (Figure 14-11, “Decision Trace Input Parameters” on page 202)



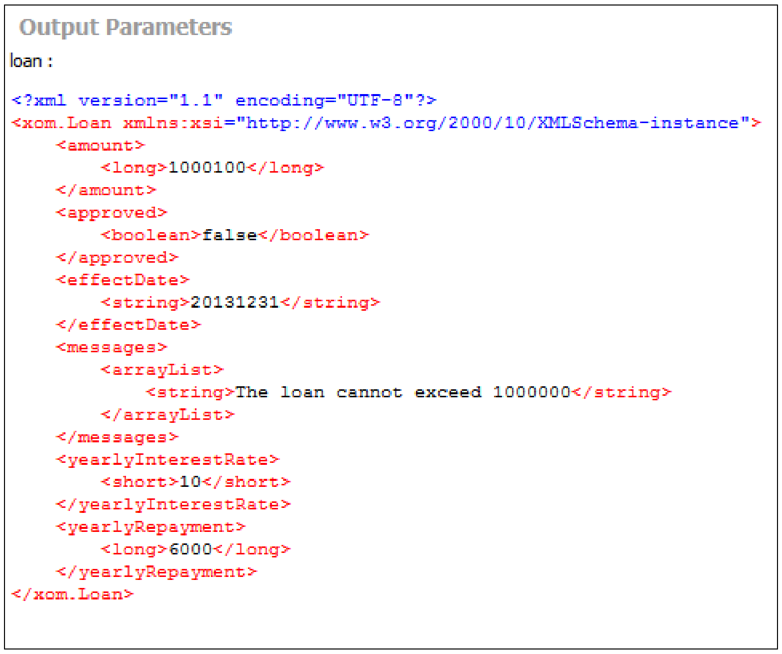
Input Parameters

borrower :

```
<?xml version="1.1" encoding="UTF-8"?>
<xom.Borrower xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
    <age>
        <short>32</short>
    </age>
    <creditScore>
        <long>600</long>
    </creditScore>
    <name>
        <string>Michelle</string>
    </name>
    <yearlyIncome>
        <long>80000</long>
    </yearlyIncome>
</xom.Borrower>
```

*Figure 14-11 Decision Trace Input Parameters*

d. Output Parameters (Figure 14-12)



Output Parameters

loan :

```
<?xml version="1.1" encoding="UTF-8"?>
<xom.Loan xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
    <amount>
        <long>1000100</long>
    </amount>
    <approved>
        <boolean>false</boolean>
    </approved>
    <effectDate>
        <string>20131231</string>
    </effectDate>
    <messages>
        <arrayList>
            <string>The loan cannot exceed 1000000</string>
        </arrayList>
    </messages>
    <yearlyInterestRate>
        <short>10</short>
    </yearlyInterestRate>
    <yearlyRepayment>
        <long>6000</long>
    </yearlyRepayment>
</xom.Loan>
```

*Figure 14-12 Decision Trace Output Parameters*

15

# Configuring the Rule Execution Servers for IBM z/OS console with virtual IP addressing

This chapter describes the use of a virtual IP address (VIPA) to allow zRules consoles to manage Rule Execution Servers for z/OS (zRES) on multiple logical partitions (LPARs) or multiple systems. This allows for uninterrupted rules deployment to continue during LPAR maintenance or downtime.

This chapter uses a scenario that consists of two LPARs, six rule execution environments, two zRES consoles, and one database, as shown in the diagram in Figure 15-3 on page 208.

The following topics are covered in this chapter:

## 15.1 Overview of a multiple LPAR environment

This environment is an Operational Decision Manager for z/OS system that consists of multiple zRES instances, which reside on multiple LPARs that are supported by a single zRules console that is connected to a single database. The ability to publish changed rules applications to a running zRES without having to restart the server, which is called *hot deployment* (15.1.1, "Hot deployment of rules in Operational Decision Manager" on page 204), from the zRules console is supported by using an internal asynchronous "publish/subscribe" notification mechanism.

An example topology is a configuration that consists of two LPARs, LPAR A and LPAR B, which reside in PLEX 1**,** as shown in Figure 15-1.

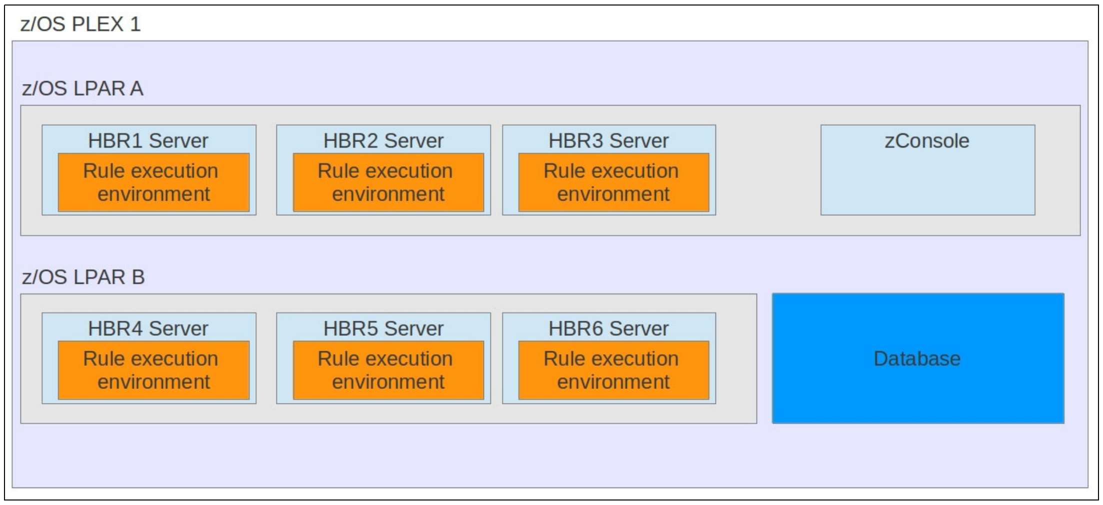

z/OS PLEX 1
z/OS LPAR A
HBR1 Server
Rule execution environment
HBR2 Server
Rule execution environment
HBR3 Server
Rule execution environment
zConsole
z/OS LPAR B
HBR4 Server
Rule execution environment
HBR5 Server
Rule execution environment
HBR6 Server
Rule execution environment
Database

*Figure 15-1 Multiple LPAR Operational Decision Manager system*

There are two styles of rules application deployment:

- Hot deployment of rules
- Cold deployment of rules

### 15.1.1 Hot deployment of rules in Operational Decision Manager

*Hot deployment* of rules in Operational Decision Manager is the ability to publish changed rules applications to a running Rule Execution Server without having to restart the server. The deployment feature works by deploying the Rule Application using the zRules console.

Rule deployment can be performed directly from the zRules console, Rule Designer, Rule Team server, or Decision Center Business console. The zRules console uses asynchronous messages to notify all rule execution environments that are registered to that zRules console. The next invocation of that rule causes the zRES to go to the database to load the newest level of that rule application.

After the zRES reads the rules from the database, it continues to use that version of the rules until it is notified of another update. Figure 15-2 shows information flows that describe how deployment, storage, registration, and notification occur.

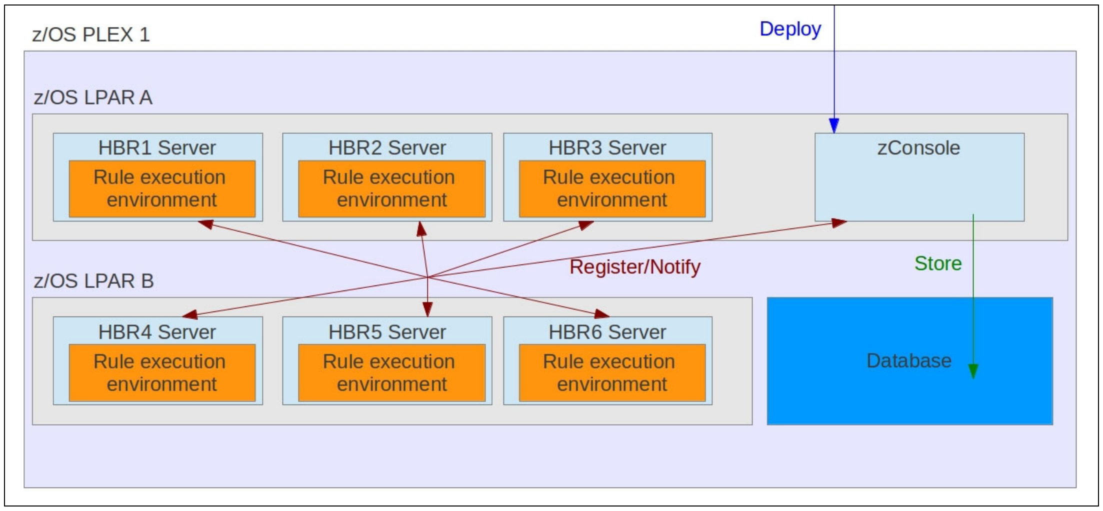

z/OS PLEX 1
Deploy
z/OS LPAR A
HBR1 Server
Rule execution environment
HBR2 Server
Rule execution environment
HBR3 Server
Rule execution environment
zConsole
Register/Notify
Store
z/OS LPAR B
HBR4 Server
Rule execution environment
HBR5 Server
Rule execution environment
HBR6 Server
Rule execution environment
Database

*Figure 15-2 Hot deployment in a multiple LPAR environment*

### 15.1.2 Cold deployment of rules in Operational Decision Manager

*Cold deployment* is performed by using a deployment method that *directly* updates the rules that are stored in the database. The rule execution environments load rules from the database, which means that on the first invocation of a ruleset, the rule execution environment goes to the database and retrieves the latest copy of the ruleset. On any subsequent invocations, it reuses the *same* ruleset *unless* it is notified of an updated version. To refresh rulesets in the rule execution environment to pick up ruleset changes, the rule execution server must be restarted.

## 15.2 Using virtual IP addressing to allow more than one zRules console to be used

Any zRES that is on the same LPAR as the zRules console can restart the zRules console, if the console fails. However, if the LPAR that hosts the zRules console fails, there is no means for the remaining zRESs to start (or restart) a zRules console. Therefore, hot deployment is not available until the LPAR that hosts the zRules console is brought back online.

This section explains the setup and behavior of a zRules console that uses VIPA to manage zRESs on multiple LPARs (or systems), therefore allowing continued hot deployment during LPAR maintenance or downtime.

The following sections describe the various elements that are associated with the loss of an LPAR that hosts a zRES and a ZRules console:

- 15.2.1, “What happens if the LPAR that hosts the zRules console fails” on page 206
- 15.2.2, “Using virtual IP addressing” on page 206
- 15.2.3, “How VIPA maintains hot deployment” on page 207

## 15.2.1 What happens if the LPAR that hosts the zRules console fails

If the LPAR that hosts the zRules console fails, for example, LPAR A in Figure 15-2 on page 205, there is no means for the remaining zRES, which is on LPAR B, to start (or restart) the zRules console (zConsole). Therefore, the hot deployment of rules is available until LPAR A is brought back online.

The rule execution environments on LPAR B still continue to execute the rules from the database. Rule deployment *is* still possible but only by using a cold deployment method because hot deployment is unavailable.

## 15.2.2 Using virtual IP addressing

Hot deployment can be more flexible in a production environment by using virtualization to allow the zRules console to be run on more than one LPAR. This situation is possible because all of the zRules console communication is performed by using TCP/IP.

To use a VIPA, define a virtual host that maps to LPAR A and LPAR B, which allows both LPARs to share the zRules console communication port and the zRules console HTTP port.

**Note:** Do not use a balancing algorithm on the port sharing because you are not sharing the load.

Send all requests to one server until it becomes unavailable. Therefore, for the example that is used here, all traffic is sent to LPAR A until the connection to LPAR A is lost, and then all traffic is sent or redirected to LPAR B.

There are many ways in which VIPA can be set up for a configuration. The following IBM publication, *z/OS Communications Server: IP Configuration Guide,* SC31-8775-20, provides all the required information to decide how to set up VIPA:

http://www-01.ibm.com/support/knowledgecenter/SSLTBW_1.13.0/com.ibm.zos.r13.halz002/f1a1b3b1151.htm%23wq464

Example 15-1 and Example 15-2 are extracts from the TCP PARMS data set from each LPAR. You can use these extracts for an example setup, and they can help you with several default settings.

*Example 15-1 TCP PARMS for LPAR A*

```
VIPADYNAMIC
  VIPADEFINE MOVE IMMED 255.255.255.252 9.20.9.53
    VIPADISTRIBUTE 9.20.9.53 PORT
  24159 34159 44159
  DESTIP ALL
ENDVIPADYNAMIC
```

*Example 15-2 TCP PARMS for LPAR B*

```
VIPADYNAMIC
 VIPABACKUP 100 MOVE IMMEDIATE 255.255.255.252 9.20.9.53
ENDVIPADYNAMIC
```

In Example 15-1 on page 206 and Example 15-2 on page 206, the following entries are underlined twice:

- `9.20.9.53`: IP address for VIPA
- `24159`: `SSPPORT` port number (not required for VIPA support)
- `34159`: `CONSOLEPORT` port number
- `44159`: `CONSOLECOM` port number

Descriptions of the ports are in Table B-1 on page 232.

The "share port" setup is also explained in the IBM publication, *z/OS Communications Server: IP Configuration Guide*, SC31-8775-20. The configuration that is shown in Example 15-3 is created by adding the following configuration to the TCP PARMS data set on both LPARs.

*Example 15-3 TCP PARMS for LPAR A and LPAR B*

```
IPCONFIG DYNAMICXCF 192.168.x.x 255.255.255.0 1
PORT
34159 TCP OMVS SHAREPORT ; zRules Console Port
44159 TCP OMVS SHAREPORT ; zRules Console ComPort
```

After the VIPA and share port are set up, the zRES needs to use the virtual host. You must modify the two zRES's parms members. In the ++HBRWDS++.SHBRPARM(HBRCMMN) member, modify the **`HBRCONSOLECOMPORT`** and **`HBRCONSOLECOMHOST`** parameters to use the virtual host and share port, as demonstrated in Example 15-4.

*Example 15-4 HBRCMMN parms for each zRES*

```
HBRCONSOLECOMPORT=44159
HBRCONSOLECOMHOST=zodm.hursley.ibm.com
```

In the ++HBRWDS++.SHBRPARM(HBRCNSL) member, modify the **`HBRCONSOLEPORT`** parameter to the shared port that is defined as shown in Example 15-5.

*Example 15-5 HBRCNSL parm for each zRES*

```
HBRCONSOLEPORT=34159
```

After these modifications, you must restart all zRESs.

## 15.2.3 How VIPA maintains hot deployment

This section describes the use of VIPA in the scenario that is presented in Figure 15-1 on page 204. The following modes of operation are described:

- "Normal operation"
- "Failure of LPAR A" on page 209
- "Restoration of LPAR A" on page 210
- "Return to the normal operating environment" on page 210

### Normal operation

With all the servers started, by using the VIPA and shared ports, the zRules console works normally. All traffic is routed to the zRules console on LPAR A, and therefore, all servers are registered to the one zRules console on LPAR A.

*Share port* enables the two zRules console address spaces to start and bind to the port. See Figure 15-3, which depicts an Operational Decision Manager for z/OS configuration using VIPA.

The following four actions are performed for the hot deployment feature:

- Deployment
- Storage
- Registration
- Notification

The registration is performed during zRES start. If the connection is lost, the zRES attempts a reconnection. If the connection attempt fails, the zRES attempts to reconnect every 10 seconds.

Deployment uses the *virtual host* and the *shared console port*. The zRules console, zConsole, stores rulesets in the database and notifies all connected rule execution environments, again, as though VIPA is not being used.

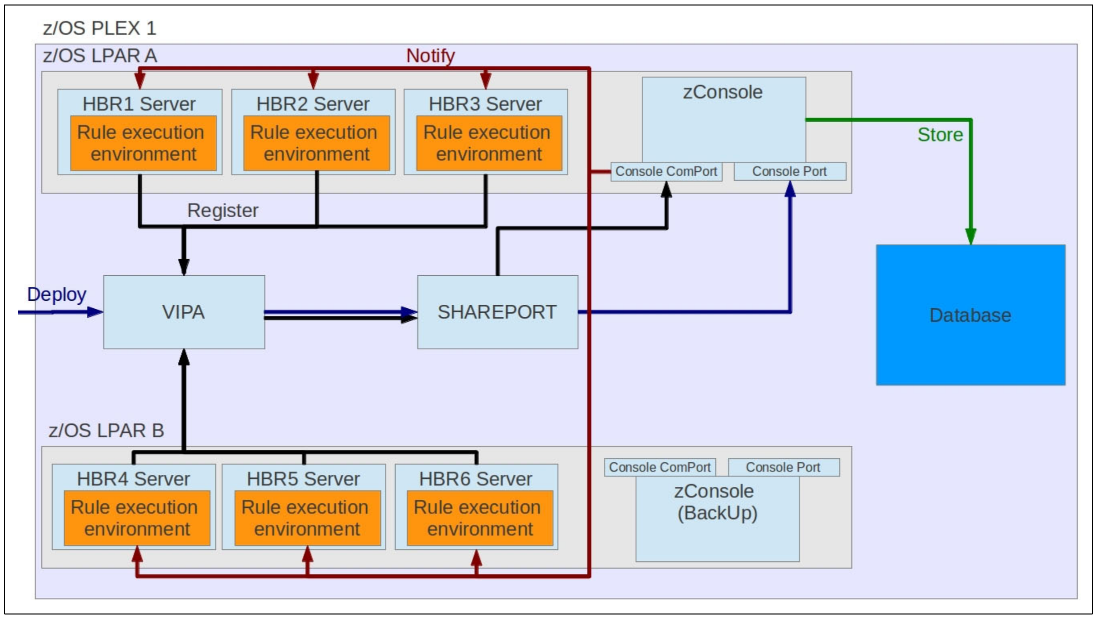


*Figure 15-3 Configuration of VIPA using two zRules consoles*

## Failure of LPAR A

This section describes the sequence of events if a failure occurs that results in the loss of access to LPAR A. The scenario is depicted in Figure 15-4.

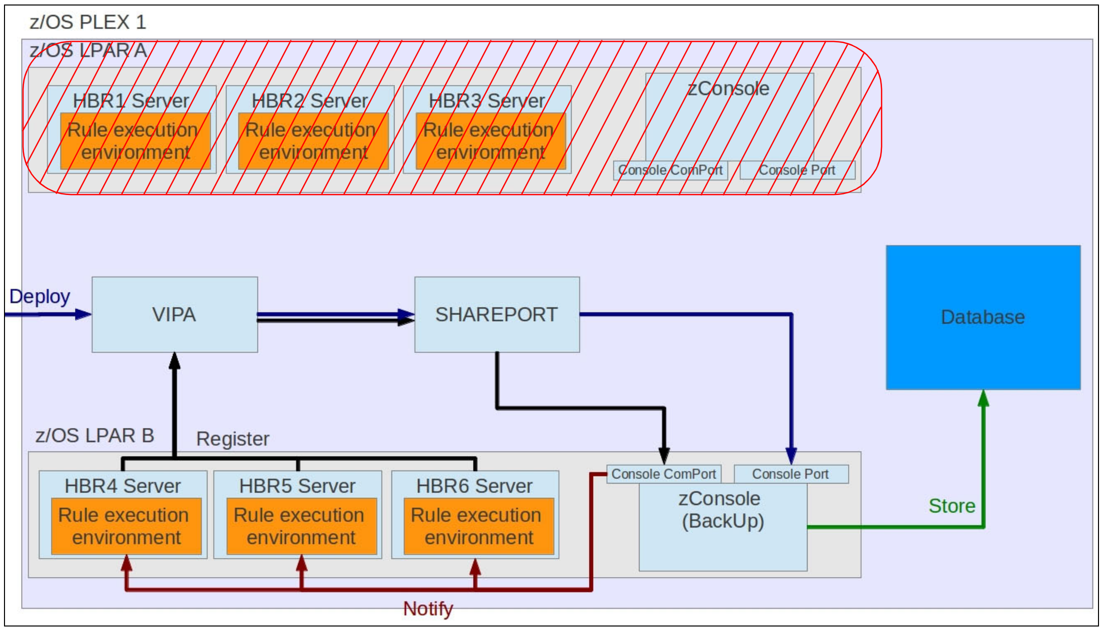


*Figure 15-4 Failover to zConsole on LPAR B*

The events occur in this sequence:

1. LPAR A fails.
2. VIPA routes all traffic that is aimed at the zRules console to the zRules console on LPAR B, which is zConsole (BackUp).
3. The rule execution environments, HBR4, HBR5, and HBR6, lose their connection to the LPAR A zRules console (zConsole) and try to reconnect.
4. The result of the reconnection attempt is an *almost instantaneous* connection to the zRules console that runs on LPAR B zConsole (BackUp), which ensures that the hot deployment can continue.

Standard rule execution and rule deployment continue from this point.

**Important:** There is no function to make the zRules console on LPAR B aware of any rule changes that are made by the zRules console that runs on LPAR A.

To work around the issue of this lack of awareness of rule changes that are made by another zRules console, manually refresh the zRES console on LPAR B, which forces new rules to be visible.

To manually refresh the zRES console on LPAR B, click **Update RuleApps** on the Explorer tab in the zRules console, as shown in Figure 15-5.

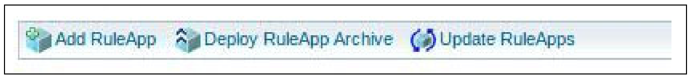


*Figure 15-5 Update RuleApps on zRules console Explorer tab*

## Restoration of LPAR A

This section describes the scenario that is associated with the restoration of a failed LPAR (LPAR A in this example) that hosts the Operational Decision Manager execution environments and a zRES console.

The failed LPAR needs to be brought online normally, and Operational Decision Manager rule execution environments also need to be started normally.

**Important:** The configuration will *not* automatically return operation to the zRules console on LPAR A after LPAR A is restored.

Traffic continues to be routed to zConsole (BackUp) on LPAR B, uninterrupted, until an intervention occurs. See Figure 15-6.

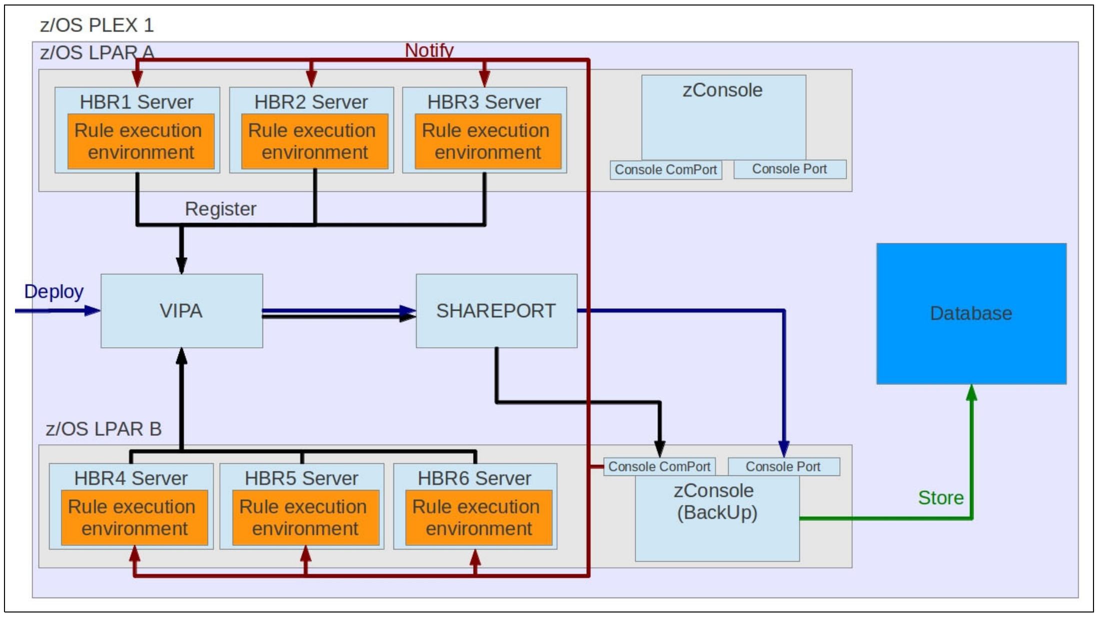


*Figure 15-6 LPAR A restored*

## Return to the normal operating environment

To return the Operational Decision Manager system to the original mode of operation, which is routing traffic to LPAR A, the zRules console that runs on LPAR B, zConsole (BackUp), needs to be restarted.

**Note:** By using this example, *fail back* is a deliberate act after LPAR maintenance or recovery is completed successfully. The behavior can be tailored in the VIPA setup to meet other configuration requirements.

The result of restarting the LPAR B zRES console, zConsole (BackUp), is shown in Figure 15-7.

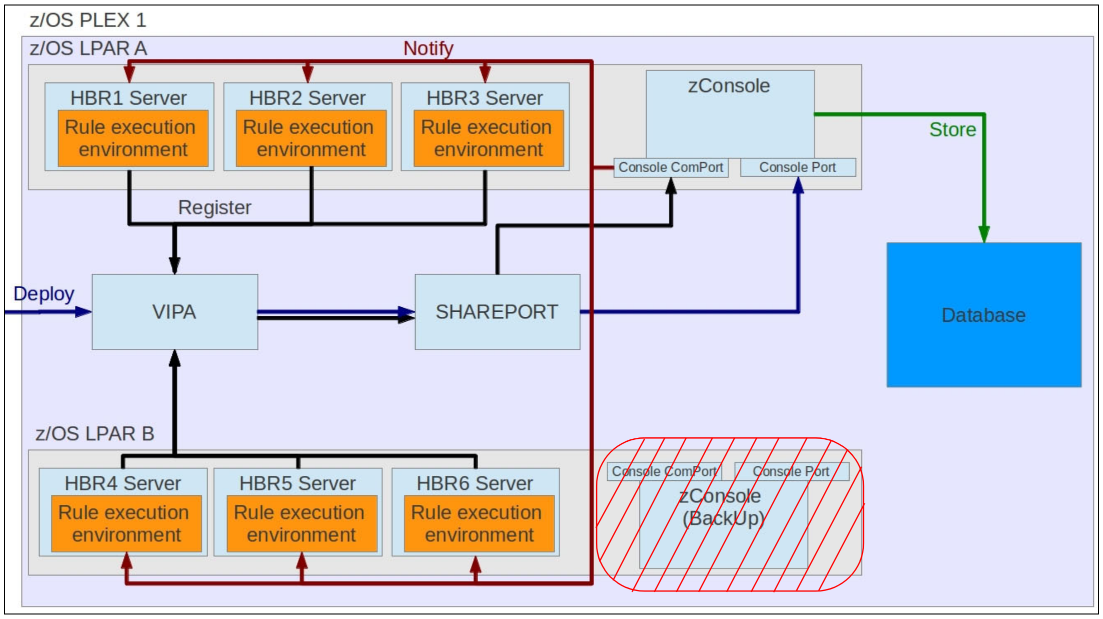


*Figure 15-7 Operational Decision Manager system following a restart of zConsole (BackUp)*

The result of restarting the zConsole (BackUp) is similar to the situation that is described in “Failure of LPAR A” on page 209:

1. VIPA routes all traffic that is aimed at a zRules console to the zRules console on LPAR A, zConsole.
2. The rule execution environments, HBR1, HBR2, and HBR3, lose their connection to the LPAR B zRules console, zConsole (BackUp), and try to reconnect.
3. The result of the attempt to reconnect is an almost instantaneous connection to the zRules console (originally used before the LPAR failure) that runs on LPAR A zConsole.

Hot deployment continues to be possible. A refresh might be required, as shown in Figure 15-5 on page 209. This time, you update RuleApps on the zRules console Explorer tab on the LPAR A zRES console, zConsole.

16

# Configuring Operational Decision Manager to collect execution data using SMF

This chapter describes how Operational Decision Manager uses the System Management Facilities (SMF) to record statistics about rule executions and how to configure SMF usage with Operational Decision Manager.

The following topics are covered in this chapter:

## 16.1 Overview

System Management Facilities (SMF) is the standard mechanism for recording all aspects of system activity on z/OS. It is used for used for accounting, performance monitoring, capacity planning, and so on. SMF recording is record based: Each application that wants to use SMF is responsible for defining their own record format, which describes the activity for that application. z/OS provides an API that allows these records to be output to either a Virtual Storage Access Method (VSAM) data set or SMF logstream. z/OS also provides programs to read back the records for producing reports. The format of all SMF records is published and users can choose how they output such reports in a human readable way. z/OS provides the infrastructure for writing and storing the records and reading them back, while application developers supply the record content.

Every SMF record has a standard common header followed by an application defined header. The common header contains a time stamp that records the time when the record was written to SMF. When reading SMF records back for formatting, a time range may be supplied to limit the records returned to just that range. All records have a type where types 0 - 127 are reserved for IBM applications.

Records can either be written as soon as events occur, or they can be buffered by the application and written out at periodic intervals. If an application chooses periodic intervals, SMF provides a system-wide interval time, which the application may interrogate.

## 16.2 Operational Decision Manager use of SMF

Operational Decision Manager support for SMF has progressed through three stages:

- In the beginning, there was support for SMF “Type 89” records. This is a generic type of record that just records the application CPU time as a whole.
- As some customers were starting to implement inter-departmental charging of their Operational Decision Manager usage, they felt the need for more detailed accounting of rule usage. Release 8.5 of Operational Decision Manager (then named WebSphere Operational Decision Manager) added support for its own type of SMF record, detailing how many times a particular ruleset was called in a given interval. These are known as *Type 120 Subtype 100 records*.
- With release 8.7 of Operational Decision Manager, the Type 120 Subtype 100 record was expanded to include the actual CPU time for each ruleset.

This chapter describes the newer Type 120 records only.

Because SMF top-level record types are limited to numbers 0 - 127 for IBM products, all these numbers are already assigned and Operational Decision Manager does not have its own top-level type. Instead, it extends another existing type, number 120, which is owned by WebSphere Application Server. Type 120 records in turn can have a number of subtypes and Operational Decision Manager is allocated subtype 100 - 199 for its records.

While the main attraction for customers of SMF with Operational Decision Manager is for charging purposes, SMF records are of interest also for profiling and monitoring. This is because it gives you an easy way to see which rulesets are being called most often and, which are consuming the most CPU.

Operational Decision Manager uses the interval mechanism of SMF recording: Statistics about rulesets are updated internally after every ruleset execution and then output to SMF at the end of each standard SMF interval. After writing the SMF records, the stats are reset to zero, ready for the next interval.

If Operational Decision Manager is stopped before the interval occurs, the record is written at the time of shutdown so that no data is lost.

## 16.3 Record format

The SMF records written by Operational Decision Manager (ODM) have the following format:

| |
|---|
| SMF Header |
| ODM Header |
| ODM exec segment |
| ODM exec segment |
| ... |

**Note:** If multiple Operational Decision Manager servers are in use on the same machine, they each output their own set of SMF header/ODM header/ODM exec segments.

### 16.3.1 SMF header

The SMF header is a standard header common to all SMF records and has the following fields:

```
typedef struct {
    char SMF120FLG;     /* System indicator          */
    char SMF120RTY;     /* Record type 120 (x'78')   */
    char SMF120TME[4];  /* Time when SMF moved record*/
    char SMF120DTE[4];  /* Date when SMF moved record*/
    char SMF120SID[4];  /* System ID                 */
    char SMF120SSI[4];  /* Not used by ODM           */
    uint16_t SMF120STY; /* Record Subtype            */
    uint16_t SMF120HDV; /* ODM record version number */
    uint32_t SMF120HDO; /* Header offset in a record */
    uint16_t SMF120HDL; /* Header length             */
    uint16_t SMF120HDN; /* Number of headers         */
} SMFRecordHeader;
```

Operational Decision Manager sets SMF120RTY to 120 to indicate a WebSphere Application Server type 120 record. SMF120STY is set to 100 to indicate that it is an Operational Decision Manager record and SMF120HDV is set to version 3.

**Note:** The Operational Decision Manager record versions are not compatible with an earlier version. Therefore, if you attempt to format a version 3 record with code written for version 2, you get abnormal results.

The SMF header is immediately followed by a single ODM header, so SMF120HDO is the size of the above SMFRecordHeader structure. SMF120HDL is the length of the ODM header. And SMF120HDN is set to 1.

## 16.3.2 ODM header

The ODM header is unique to the subtype 100 records and has the following structure:

```
typedef struct {
    char SMF120VER[16]; /* ODM version                    */
    char SMF120XUL[32]; /* XU Location                    */
    char SMF120XUT[32]; /* XU Type                        */
    char SMF120SDT[16]; /* SMF interval start date        */
    char SMF120STM[16]; /* SMF interval start time        */
    char SMF120EDT[16]; /* SMF interval end date          */
    char SMF120ETM[16]; /* SMF interval end time          */
    uint32_t SMF120EXO; /* Exec section start point       */
    uint16_t SMF120EXL; /* Length of an exec segment      */
    uint16_t SMF120EXN; /* Total number of exec segments  */
} HBRSMF120ST100RecordHeader;
```

- SMF120VER is the version of ODM, for example, 8.7.1.0
- SMF120XUL is the execution unit (XU) location. This is a string that identifies the server that the rule engine is running in:
  - For zRule Application Server for z/OS (zRES), this is the SSID of the master address space
  - For CICS, it is the application identifier (APPLID)
  - For WebSphere Application Server, it takes the form “cell_name:node_name:server_name”
  - For zRES embedded, it is the batch job name
  - For Java batch, it is the class name
- SMF120XUT is a string identifying the type of server in which the XU is running:
  - For zRES, this is “zRule Execution Server”
  - For CICS JVM, this is “CICS JVM Server” (note that CICS can be configured to use either zRES or the CICS JVM)
  - For zRES embedded, this is “Embedded zRule Execution Server”
  - For WebSphere Application Server, this is “WebSphere Application Server”
  - For Java batch, this is “Embedded Rule Execution Server”
- SMF120SDT, SMF120SDM, SMF120EDT, and SMF120EDM denote the interval start and end date and time

Immediately following the ODM header are zero or more exec segments. SMF120EXO is equal to the length of the SMF header plus the length of the ODM header. SMF120EXL contains the length of each exec segment and SMF120EXN contains the number of exec segments.

**Note:** The number of exec segments can be zero if “HBRSMF120ST100EMPTY” is set to `YES` to request that records are written even if there was no activity in the interval.

### 16.3.3 ODM exec segment

One ODM exec segment is output for each unique ruleset that is executed during the preceding interval. It has the following structure:

```
typedef struct {
    uint64_t RULEXNUM;   /* Ruleset successful execution count      */
    uint64_t RULEXBAD;   /* Ruleset failed execution count          */
    uint64_t RULEXFSUM;  /* Ruleset sum of fired rules              */
    uint64_t RULEXCALLS; /* Ruleset number of zRES calls            */
    uint64_t RULEXTIME;  /* Ruleset zRES total elapsed Java time    */
    uint64_t RULEXTMAX;  /* Ruleset zRES maximum elapsed Java time  */
    uint64_t RULEXTMIN;  /* Ruleset zRES minimum elapsed Java time  */
    uint64_t RULEXCPU;   /* Ruleset zRES total CPU Java time        */
    uint64_t RULEXCMAX;  /* Ruleset zRES maximum CPU Java time      */
    uint64_t RULEXCMIN;  /* Ruleset zRES minimum CPU Java time      */
    char RULEXPATH[256]; /* Ruleset execution path                  */
} HBRSMF120ST100RecordExec;
```

► **Note:** All time values are in microseconds.

- RULEXNUM contains the number of successful executions for the given execution path in the preceding interval. This normally is the same value as RULEXCALLS.
- RULEXBAD contains the number of failed executions. A ruleset call can fail if for instance there is an error in the rule logic or if an attempt is made to access a null variable.
- RULEXFSUM is the total number of rules that fired.
- RULEXCALLS is the total number of calls made to execute the given ruleset. This is equal to RULEXNUM + RULEXBAD.
- RULEXTIME is the total wall clock time from entry to ODM/Java to exit for this ruleset.
- RULEXTMAX is the maximum wall clock time taken by a single rule execution for this ruleset in this interval. Take this figure lightly because it is affected by what else is happening on the machine, for instance, a Garbage Collection (GC) cycle or just that there are many competing threads at that particular time.
- RULEXTMIN is the minimum wall clock time.
- RULEXCMAX and RULEXCMIN are similar to RULEXTMAX and RULEXTMIN except that they measure CPU time for the thread executing the rule. The max figure again can be skewed by a GC cycle because the current thread participates in the GC work (this is just how Java works).
- RULEXPATH is the ruleset execution path. There is only one exec segment per RULEXPATH for a given interval/XU.

## 16.4 Implementation

The Operational Decision Manager SMF records record only the time spent in Java code from the point where the Operational Decision Manager Java code got called to the point where it returned. No attempt is made to record native JNI CPU time before or after the call because it is considered that this will be tiny in comparison to the Java time.

In general, the way that it works is as follows:

- On entry to the Operational Decision Manager Java code, the current wall clock and thread CPU time are saved in a local variable
- The rule parameters are unmarshalled and a call made to execute the ruleset
- As part of this execution, an internal record is updated with the number of fired rules, and so on, and this record is stored in a hashtable keyed by the ruleset path
- When the call to execute the ruleset returns, just before the return from the Operational Decision Manager Java code, the difference between the current and saved wall clock and thread CPU time is calculated and the internal record updated with these values
- Meanwhile, a background thread sleeps for the SMF interval time. When it wakes up, it gets the records from the hashtable, writes them out to SMF and then clears the hashtable ready for the next interval

The thread CPU time is obtained using the TIMEUSED macro. The documentation for that macro includes the following note.

**Note:** TIMEUSED returns normalized CPU time. Some servers are configured with z Systems Application Assist Processors (zAAPs) or IBM z Systems Integrated Information Processors (zIIPs), which run at a faster speed than the normal CP processors. In this case, zAAP time and zIIP time are normalized to the equivalent time that it takes to run on a normal CP when accumulated into total CPU time.

## 16.5 Configuration

The configuration steps vary according to which XU is being used, but in all cases the server must have at least READ access to the RACF BPX.SMF profile of the FACILITY class.

SMF recording is turned off by default so the following steps must be taken to enable it:

- For zRES, set the property HBRSMFST100=YES in SHBRPARM(HBRMSTR). Optionally, set HBRSMFST100EMPTY=YES if empty SMF records are required to be output even when there is no activity during the interval.
- For CICS, set the same properties as shown in the preceding bullet point but in the member SHBRPARM(HBRCICSJ) for the CICS region.
- For WebSphere Application Server, it is a little more complicated. See the IBM Knowledge Center for the detailed instructions, but in summary, follow these steps:
  - Add {pluginClass=com.ibm.rules.hbr.smf.SMFPlugin,allowEmpty=NO} to the plug-ins property for the XU (or allowEmpty=YES if empty SMF records are required)
  - Add the Operational Decision Manager library to the LIBPATH for the JVM
  - Add HBRSMFST100=YES to the JVM system properties
- For Java batch, set the property HBRSMFST100=YES in SHBRPARM(HBRBATCH), then run the HBRCJCFG JCL to generate the ra.xml

Operational Decision Manager also reads the standard SMF INTVAL and SYNCVAL values to decide at what time to output the SMF record. These are global SMF parameters defined in the SMFPRMxx parmlib member. SYNCVAL defines what minute of the hour the intervals are relative to and is normally set to zero. INTVAL defines the length of the SMF interval in minutes. Operational Decision Manager uses the SMFINTVL macro to read the values of INTVAL and SYNCVAL both at start and every time the writer thread wakes up to write the next record. Therefore, it is not necessary to restart Operational Decision Manager if the INTVAL is changed.

## 16.6 Troubleshooting

Operational Decision Manager outputs various log messages, which may be useful in diagnosing issues with SMF. You need HBRTRACELEVEL set to the appropriate level to see these messages.

### INFO: GBRZP8013I SMF recording is enabled

If you see this message, you know that SMF is enabled and everything is alright.

### INFO: GBRZP8014I SMF INTVAL is 10 minutes and SYNCVAL is 0 minutes

This message provides confirmation of the INTVAL and SYNCVAL settings.

### SEVERE: GBRZP8012E An error occurred while attempting to locate libhbrsmf.so

Operational Decision Manager requires a native JNI library. And if you see this message, it means that it has failed to find it, meaning that the LIBPATH is set incorrectly. This should not happen for zRES and CICS because the LIBPATH is set automatically at installation time. However, you might see this under WebSphere Application Server if the manual step to set the LIBPATH is missed or done incorrectly.

### SEVERE: GBRZP8001E The calling process is not permitted to the BPX.SMF facility class

This means that the owner of this address space does not have the required RACF privileges to write SMF records. Check that the configuration step is done to give the server READ access to the BPX.SMF profile.

Additionally, if HBRTRACELEVEL is set to `FINE` or `ALL`, the following messages can also be seen.

### FINE: SMF Plugin rule '/MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0' elapsed time 6223us, cpu time 6202us

You can use this message to see the time taken for an individual ruleset invocation. The times are in microseconds and the path varies according to the ruleset invoked.

### FINE: SMF Writer successfully wrote SMF record

You see this message at the point where the SMF record is written.

## 16.7 Formatting SMF output

After configuring SMF and running some rule executions to produce SMF data, how do you view the output? The format of the SMF records is published (see the structures referred to earlier in this chapter, for example, SMFRecordHeader) so you can use the IFASMFDP program with your own formatter if you want. However, most people use the IBM supplied sample programs, HBRSMFC and HBRSMFP. The code for these programs is provided if you want to tweak the output.

The way these programs work is that the JCL is split into two parts. The first calls IFASMFDP to extract the SMF data and the second part runs the formatting program on the extracted data. For example:

```
//SMFDUMP  EXEC PGM=IFASMFDP
//DUMPINA  DD   DSN=SYS1.MVTC.MANA,DISP=SHR,AMP=('BUFSP=65536')
//DUMPINB  DD   DSN=SYS1.MVTC.MANB,DISP=SHR,AMP=('BUFSP=65536')
//DUMPOUT  DD   DISP=(NEW,PASS),DSN=&&SMFDMP
//SYSIN  DD *
  INDD(DUMPINA,OPTIONS(DUMP))
  INDD(DUMPINB,OPTIONS(DUMP))
  OUTDD(DUMPOUT,TYPE(120))
  START(0000)
  END(2359)
/*
```

Here, the IFASMFDP program is being called to extract the data. DUMPINA and DUMPINB are the system-dependent SMF data sets (see your system administrator for details). The START and END fields denote the start and end times that are required. Here, we are requesting all records. The extracted data is output to the temporary data set denoted by OUTDD -> DUMPOUT.

```
//SMFPRINT EXEC PGM=HBRSMFP,REGION=0M
//STEPLIB  DD DISP=SHR,DSN=++HBRHLQ++.SHBRLOAD
//SMFIN    DD DISP=SHR,DSN=*.SMFDUMP.DUMPOUT
```

In the second step, we run the IBM supplied HBRSMFP program to format the SMF records. This reads from the SMFIN data set as follows (code simplified for clarity):

```
SMFIN = fopen("dd:SMFIN", "rb, type=record");
for (;;) {
     n = fread(smf, 1, 32756, SMFIN);
     if (n == 0) break;
     if (smf->SMFHeader.SMF120RTY == 120 && smf->SMFHeader.SMF120STY == 100 &&
smf->SMFHeader.SMF120HDV == 3) {
          printSMF120ST100(smf);
     }
}
```

This C code opens the SMFIN data set for binary reading in record-based format and then processes each entry one record at a time. 32,756 is the maximum length of an SMF record. (There is no support for Operational Decision Manager SMF records greater than 32,756 in length). The “if” statement checks that this is an SMF Type 120 record, with subtype 100 and version 3. All other records are ignored. If the record matches those criteria, it is printed.

HBRSMFC outputs each record in comma-separated value (CSV) format. The output from HBRSMFP is more appealing and looks like this:

```
*******************
* SMFRecordHeader *
*******************
SMF120RTY = 120
SMF120SID = MVGA
SMF120STY = 100
SMF120HDV = 3
```

```
SMF120HDO = 32
SMF120HDL = 152
SMF120HDN = 1
******************************
* HBRSMF120ST100RecordHeader *
******************************
SMF120VER = 8.7.1.0
SMF120XUL = HBRS
SMF120XUT = zRule Execution Server
SMF120SDT = 11/18/14
SMF120STM = 1:40:00 PM
SMF120EDT = 11/18/14
SMF120ETM = 2:30:00 PM
SMF120EXO = 188
SMF120EXL = 672
SMF120EXN = 2
****************************
* HBRSMF120ST100RecordExec *
****************************
RULEXNUM = 3
RULEXBAD = 0
RULEXFSUM = 3
RULEXCALLS = 3
RULEXTIME = 1512617
RULEXTMAX = 1512073
RULEXTMIN = 252
RULEXCPU = 58826
RULEXCMAX = 58299
RULEXCMIN = 252
RULEXPATH = /reszminiloanApp/1.0/reszminiloan/1.0
****************************
* HBRSMF120ST100RecordExec *
****************************
RULEXNUM = 603468
RULEXBAD = 0
RULEXFSUM = 403467
RULEXCALLS = 603468
RULEXTIME = 23029544
RULEXTMAX = 19923
RULEXTMIN = 27
RULEXCPU = 22772739
RULEXCMAX = 13391
RULEXCMIN = 27
RULEXPATH = /MiniLoanDemoRuleApp/1.0/MiniLoanDemo/1.0
```

Most of the fields have already been described. This particular run was done on a system with INTVAL set to 10 minutes. It might look odd that there is a 50-minute gap between SMF120STM and SMF120ETM. But that is because in this case although the server was started shortly after 1:40, there was no activity to record until the 10-minute interval leading up to 2:30. If HBRSMFST100EMPTY=YES had been set, empty records with SMF120EXN=0 would have been output every 10 minutes.

There are two exec segments here, one for each of the rulesets executed in this 10-minute period. The reszminiloanApp was called three times and MiniLoanDemoRuleApp 603,468 times. The RULEXTMAX value for reszminiloanApp might look unusually large, but because that was run first, it is subject to the usual startup overheads. You can see that the actual CPU time is much smaller. In the second exec segment, RULEXFSUM is smaller than RULEXNUM. That is because for Miniloan, not every call results in a rule firing. You can also see that when the server is warmed up, nearly all of the elapsed time consists of CPU time (on a busy machine this would not be the case).

17

# Problem determination

This chapter describes some common problems with Operational Decision Manager (ODM), how to diagnose them, and if there appears to be a bug in ODM, what data to submit with a problem management record (PMR).

The following topics are covered in this chapter:

## 17.1 Performance

Here are some tips on what to look for if it is suspected that there is a performance problem:

- COBOL copybook size. This should ideally be less than about 10 k in size. If it is more than 1 MB, you likely see poor performance due to the cost of copying the parameters to and from zRule Execution Server for z/OS (zRES). Consider whether the copybook can be split so that you are only supplying the part of the copybook that is required by the decision.
- Try to avoid doing an HBRCONN for every rule execution. There is an overhead that is associated with HBRCONN and it is more efficient to make multiple HBRRULE calls for every HBRCONN.
- Java heap size. Ensure that your heap is large enough to avoid excessive Garbage Collection. Turn on -verbose:gc in your JAVA_OPTIONS and look at the trace to check your heap occupancy (not described here because there is plenty of guidance on the web).

## 17.2 MustGather

If you are unable to fix your problem and need to submit a PMR, perform the following steps to supply IBM with the data needed to diagnose the problem:

1. Set HBRTRACELEVEL to FINE in the ++HBRWORKDS++.SHBRPARM(HBRCMMN) member.
2. Stop the zRule Execution Server and restart it.
3. Re-create the problem encountered.
4. Provide all log files in the ++HBRWORKPATH++/logs directory.
5. Provide the SYSOUT, SYSPRINT, HBRPRINT, SYS00001, and SYS00002 job output elements from the SDSF job logs for both the Master and Console address spaces. In the case of an abend, also provide the SYSABEND element.
6. Provide all the members in the ++HBRWORKDS++.SHBRPARM data sets.

Part 3

# Appendixes

Part 3 includes the following appendixes:

A

# Calling out from a ruleset to a Virtual Storage Access Method file to augment data

Occasionally, you are required to call out from a ruleset to augment the data that is required to make a decision. Only call out from a ruleset to augment data in the following circumstances:

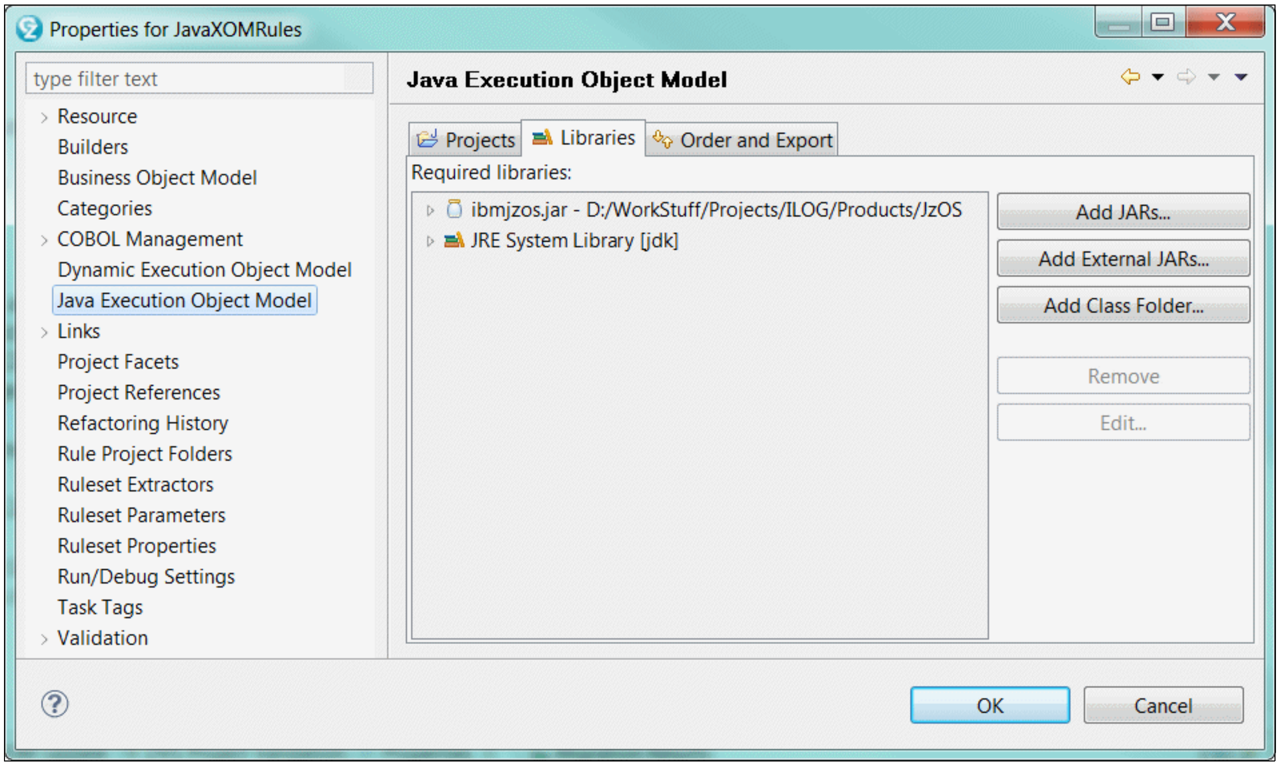
- The data is not easily accessible to the application program.
- The decision cannot be made without the data.
- The data is required only in exceptional circumstances by the decision.

The JzOS libraries that come with IBM Java Runtime Environment 6 for z/OS provide a set of classes that can be used to access a record from a VSAM file. The following code shows an example of using the JzOS library classes to read a specified record from a VSAM file. You can use this code within a business object module (BOM) method to augment the data that is available to a decision. To use the JzOS classes within a BOM method, you must first copy the JzOS library to your computer to make it available to the rule project in which you want to author your rules.

The JzOS library is in your z/OS installation directory:

`<`*`JAVA6_INSTALL_ROOT`*`>/lib/ext/ibmjzos.jar`

To allow these classes to be available to your BOM authoring, you must add this Java archive (JAR) file as an external JAR to the Java Execution Model for your project. Use the project Properties panel, as shown in Figure A-1.


Properties for JavaXOMRules

type filter text

- Resource
- Builders
- Business Object Model
- Categories
- COBOL Management
- Dynamic Execution Object Model
- Java Execution Object Model
- Links
- Project Facets
- Project References
- Refactoring History
- Rule Project Folders
- Ruleset Extractors
- Ruleset Parameters
- Ruleset Properties
- Run/Debug Settings
- Task Tags
- Validation

Java Execution Object Model

Projects | Libraries | Order and Export

Required libraries:

- ibmjzos.jar - D:/WorkStuff/Projects/ILOG/Products/JzOS
- JRE System Library [jdk]

*Figure A-1 Adding the JzOS JAR file to the project class path*

The class ReadKsdsVsam has a constructor and one method on it (Example A-1). This class is designed to read a single row from a key-sequenced data set (KSDS) format Virtual Storage Access Method (VSAM) file, based on a supplied record key, and to return the corresponding row as a byte array. By knowing the format of the record structure, the required data can then be read out from the byte array and copied into local rule variables. The byte array can be read simply by using Java substringing. Or if the record is more complex, the byte array can be read by using the Java record framework tooling. The Java record framework tooling is available in IBM Rational® Application Developer or IBM Rational Developer for System z®.

*Example A-1 ReadKsdsVsam.java*

```
import java.io.UnsupportedEncodingException;
import com.ibm.jzos.ZFile;
import com.ibm.jzos.ZFileException;
public class ReadKsdsVsam
{
   private ZFile zFile;
   private String filename;
   private String options;
   private int lrecl;
   private int keyLen;
   private byte[] keyBytes;
public ReadKsdsVsam(String filenameInput, int lreclInput, int keyLenInput) throws
ReadKsdsVsamException
{
   // Set the options to open the file as VSAM type file, read only
   options = "rb,type=record";
   this.filename = filenameInput;
   this.lrecl = lreclInput;
   this.keyLen = keyLenInput;
   // Format the file name
   if(!filename.startsWith("//"))
   filename = "//" + filename;
   try { // Check the file exists
      if(!ZFile.exists(filename)) {
         throw new ReadKsdsVsamException("File "+filename+" does not exist");
      }
         // Open the file
         zFile = new ZFile(filename, options);
      }
      catch (ZFileException zfe)
      {
         throw new ReadKsdsVsamException(zfe);
      }
   }
}
public byte[] readRecord(String key) throws ReadKsdsVsamException
{
   byte[] record = new byte[lrecl];
try {
      keyBytes = key.getBytes(ZFile.DEFAULT_EBCDIC_CODE_PAGE);
      boolean located = zFile.locate(keyBytes, 0, keyLen, ZFile.LOCATE_KEY_EQ);
      if (!located) throw new ReadKsdsVsamException("Record: "+key+" cannot be
found");
      zFile.read(record);
   }
   catch (ZFileException zfe) {
      throw new ReadKsdsVsamException(zfe);
   }
      catch (UnsupportedEncodingException uee) {
      throw new ReadKsdsVsamException(uee);
   }
   return record;
}
public void closeFile() throws ReadKsdsVsamException
{
   try {
      zFile.close();
   }
      catch (ZFileException zfe) {
      throw new ReadKsdsVsamException(zfe);
   }
}
}
```

In Example A-1 on page 228, the constructor `public ReadKsdsVsam(String filenameInput, int lreclInput, int keyLenInput) throws ReadKsdsVsamException` takes the following arguments:

| | |
|---|---|
| **filenameInput** | The fully qualified name of the file to read |
| **lreclInput** | The length of the record |
| **keyLenInput** | The length of the record key |

The constructor checks to see whether the supplied file exists and then opens the file for reading. If any of these operations fail, it throws a `ReadKsdsVsamException` with the reason for the failure. This method can be verbalized and called in the initial actions section of the ruleflow.

To read a record from the VSAM file, you then use the following method:

```
public byte[] readRecord(String key) throws ReadKsdsVsamException
```

This method takes in the key of the record to read as a String and returns the contents of the record (including the key) as a byte[] array in the code page in which it was read.

Finally, the close() method is called to close the file. This method can be verbalized and called in the final actions section of the ruleset ruleflow.

B

# Configuring runtime values by using variables defined in HBRINST

Operational Decision Manager (ODM) can be customized in the HBRINST member of the ++HBRHLQ++.SHBRPARM data set. Amending the variables in the member, when the customizing job is run, results in the values that are in HBRINST being placed into the customized members that are generated by the execution of the job.

Use the tables in the following sections, which group values into related areas, to amend the HBRINST member variables to your system environment:

- B.1, “Rules z/OS” on page 232
- B.2, “CICS” on page 233
- B.3, “CICS JVM server” on page 234
- B.4, “IMS” on page 234
- B.5, “DB2 database” on page 234
- B.6, “WebSphere Application Server” on page 236
- B.7, “WebSphere Optimized Local Adapters script parameters” on page 237
- B.8, “WebSphere Application installation script parameters” on page 237
- B.9, “Subsystem ID used by COBOL management” on page 238
- B.10, “WebSphere Application Server Liberty Profile” on page 238

For more details about the HBRINST member and other members, see the *z/OS configuration variables* topic in the IBM Operational Decision Manager IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.zos.config/topics/con_ds_jcl_and_runtime_vars.html

## B.1 Rules z/OS

Table B-1 lists the variables to create a new instance of Decision Server for z/OS.

*Table B-1 Variables for rules on z/OS*

| Variable | Example value | Description |
|---|---|---|
| *++HBRSSIDLIST++* | `HBR1,HBR2` | A zRule Execution Server for z/OS server group consisting of a list of 1 - 32 subsystem IDs separated by commas, for example: `HBR1,HBR2,HBR3`.<br><br>The first ID in the list is the primary server, from which you start the shared console. Rule execution is routed to the first available server in the list. Other servers execute rulesets only if rule execution is transferred to them. To route rule execution to a particular server, specify its ID first. |
| *++HBRHLQ++* | `HBR.V871` | High-level qualifier (HLQ) for Decision Server for z/OS data sets. This value is the installation target library from the product installation. |
| *++HBRINSTPATH++* | `/usr/lpp/zDM/V8R7M1` | This value is the root installation directory for the Operational Decision Manager product in z/OS UNIX System Services. |
| *++HBRWORKPATH++* | `/u/HBR1` | Working directory for the server instance. |
| *++HBRWORKDS++* | `HBR.WORKDS` | HLQ for the working data sets that contain customized JCL for creating an execution environment instance. |
| *++HBRJAVAHOME++* | `/java/java7_64/J7.0_64` | This value is the *root* location of the JVM on UNIX System Services. (Java 7 is recommended; Java 6.0.1 is also supported). |
| *++HBRSSPPORT++* | `24114` | Port number for a zRule Execution Server that is set up in *TESTING* mode to use. This is used in Decision Validation Service (DVS) testing. It can also be used by Decision Center for running tests or simulations against a decision service. |
| *++HBRCONSOLEPORT++* | `34114` | This value is the port that is used for the zRule Execution Server for the z/OS Execution Server Console. This value is the port that you use to deploy and view deployed artifacts. |
| *++HBRCONSOLECOMPORT++* | `44114` | This value is the port that is used by the Console and zRule Execution Server for z/OS instance to communicate with each other. |
| *++HBRCONSOLECOMHOST++* | `localhost` | Leaving this as `localhost` uses the default machine address. For a cross-LPAR setup (see a later section), this can be adjusted. |

| Variable | Example value | Description |
|---|---|---|
| ++*HBRMODE*++ | `NORMAL`, `CONSOLE`, `NATIVE`, or `TEST` | The zRule Execution Server has four possible configurations:<br>► In `NORMAL` mode, the server accepts a COBOL or PL/I workload and processes it using its internal Java virtual machine (JVM). The server settings can be changed using the console. This mode is used when executing and managing rules from a mainframe.<br>► In `CONSOLE` mode, the server provides the Rule Execution Server (RES) console web application on the ++*HBRCONSOLEPORT*++ port.<br>► In `NATIVE` mode, the server does not start a JVM to process rules.<br>► In `TEST` mode, the server provides testing and simulation execution, for Decision Server and DVS, on ++*HBRSSPPORT*++. It does not accept connections from local COBOL or PL1 clients. |
| ++*HBRLANG*++ | `En_US` | Language used by the server. The list of supported languages is in the HBRCMMN data set member. The default value is `En_US`. |
| ++*HBRTRACELEVEL*++ | `ALL`, `FINE`, `INFO`, `WARNING`, `SEVERE`, or `OFF` | Trace level during execution:<br>► `ALL`: Logs all messages, including internal traces<br>► `FINE`: Logs debug messages, informational messages, errors, and warnings<br>► `INFO`: Logs informational messages, errors, and warnings<br>► `WARNING`: Logs errors and warnings<br>► `SEVERE`: Logs errors only<br>► `OFF`: No tracing |
| ++*HBRPERSISTENCETYPE*++ | `DB2` or `FILE` | Type of persistence layer that is used to store deployed artifacts. Set this variable to `DB2` or `FILE`. |

## B.2 CICS

If you are configuring CICS to execute rules on an instance of zRule Execution Server for z/OS, customize the variables that are listed in Table B-2.

*Table B-2 Variables for CICS*

| Variable | Example value | Description |
|---|---|---|
| ++*CICSHLQ*++ | `CTS420.CICS` | This value is the HLQ for the CICS installation. Change this value to match the CICS installation HLQ. |
| ++*CICSCSDDSN*++ | `CTS420.APPLID.DFHCSD` | This value is the HLQ for the CICS region CICS system definition data set (CSD) file. For each new region into which a zRule Execution Server for z/OS is to be deployed, this value must be updated. |
| ++*CICSLIST*++ | `DFHLIST` | CICS start group list that is specified for the **`GRPLIST`** parameter. |

## B.3 CICS JVM server

If you are configuring an instance of zRule Execution Server for z/OS running on a CICS JVM server, customize the variables that are listed in Table B-3. Some of these variables are repeated from Table B-2 on page 233.

*Table B-3 Variables for CICS JVM server*

| Variable | Example value | Description |
|---|---|---|
| ++*CICSHLQ*++ | `CTS420.CICS` | This value is the HLQ for the CICS installation. Change this value to match the CICS installation HLQ. |
| ++*CICSCSDDSN*++ | `CTS420.APPLID.DFHCSD` | This value is the HLQ for the CICS region CSD file. For each new region into which a zRule Execution Server for z/OS is to be deployed, this value must be updated. |
| ++*CICSLIST*++ | `DFHLIST` | CICS start GRPLIST. List of groups containing the resource definitions that are created when you run the HBRCSD job. |
| ++*JDBCPLAN*++ | `DSNJCC` | This value is the planned use for Java Database Connectivity (JDBC) connections and CICS. |

## B.4 IMS

If you are configuring IMS to execute rules on an instance of zRule Execution Server for z/OS, customize the variables that are listed in Table B-4.

*Table B-4 Variables for IMS*

| Variable | Example value | Description |
|---|---|---|
| ++*IMSHLQ*++ | `IMS.V10.DBDC` | HLQ for the data sets of the IMS installation. |
| ++*IMSREGID*++ | `IMOA` | ID of the IMS instance to be used. |
| ++*IMSREGHLQ*++ | `IMSDATA.IMOA` | HLQ of the IMS region data sets. |

## B.5 DB2 database

If you are using a DB2 database as the persistence layer, customize the variables that are listed in Table B-5.

*Table B-5 Variables for DB2*

| Variable | Example value | Description |
|---|---|---|
| ++*DB2HLQ*++ | `SYS2.DB2.V10R1` | HLQ of the DB2 installation. |
| ++*DB2RUNLIB*++ | `DSNV10GP.RUNLIB.LOAD` | DB2 runtime library. |
| ++*DB2SUBSYSTEM*++ | `db2_subsystem_id` | DB2 subsystem name. |
| ++*DB2LOCATION*++ | `DSNV10GP` | DB2 location name that is used to connect to this DB2 subsystem. |
| ++*DB2VCAT*++ | `DSNV10GP` | DB2 integrated catalog facility (ICF) catalog. |

| Variable | Example value | Description |
|---|---|---|
| *++DB2ADMIN++* | `DB2ADMINID` | User ID that is authorized to create Events DB2 databases. |
| *++DB2SCHEMA++* | `ODMSCHMA` | The schema that is used to create and access the DB2 artifacts. |
| *++RESDATABASE++* | `RESDB1` | Name of the database that is used by the zRule Execution Server for z/OS instance. |
| *++RTSDATABASE++* | `RTSDB1` | Name of the database that is used by the Decision Center instance. |
| *++EVDATABASE++* | `EVDB1` | Name of the database that is used by the Events runtime. |
| *++RESSTOGROUP++* | `RESSTG1` | Name of the storage group that is used by the zRule Execution Server for z/OS instance. |
| *++RTSSTOGROUP++* | `RTSSTG1` | Name of the storage group that is used by the Decision Center instance. |
| *++EVSTOGROUP++* | `EVSSTG1` | Name of the storage group that is used by the Events runtime instance. |
| *++DB2TABLEBP++* | `BP1` | Buffer pool name for the tables. |
| *++DB2INDEXBP++* | `BP2` | Buffer pool name for the indexes. |
| *++DB2LOBBP++* | `BP3` | Buffer pool name for large objects. |
| *++DB2SAMPLEPROGRAM++* | `DSNTEP2` | DB2 program name. |
| *++DB2SAMPLEPROGRAMPLAN++* | `DSNTEP91` | DB2 plan name. |
| *++DB2BP4K++* | `BP4K` | Buffer pool name for 4 K objects. |
| *++DB2BP8K++* | `BP8K` | Buffer pool name for 8 K objects. |
| *++DB2BP32K++* | `BP32K` | Buffer pool name for 32 K objects. |
| *++DB2GROUP++* | `ODMGROUP` | The RACF group that is granted access across the ODM tables. Connect +*++DB2USER++* to this group. |
| *++DB2USER++* | `ODMDBUSR` | User ID for accessing the DB2 database. |
| *++DB2PSWD++* | `<your password>` | Password for accessing the DB2 database. |
| *++DB2JARLOCN++* | `/usr/lpp/db2v10/classes` | Location of the DB2 classes in UNIX System Services. |
| *++DB2NATIVELOC++* | /usr/lpp/db2v10/lib | Location of the DB2 native library files. |

## B.6 WebSphere Application Server

If you are configuring Operational Decision Manager on WebSphere Application Server for z/OS, customize the variables that are listed in Table B-6.

*Table B-6 Variables for WebSphere Application Server*

| Variable | Example value | Description |
|---|---|---|
| ++*WASINSTPATH*++ | `/WebSphere/V85IL2Z1/Appserver` | Installation directory of WebSphere Application Server. |
| ++*WAS_HOME*++ | `/WebSphere/AppServer/Profile` | WebSphere Application Server home directory. It is unique for each server instance. |
| ++*SECURITYTYPE*++ | `RACF` | Set to `RACF` if your WebSphere Application Server system is configured to use RACF. Set to `WAS` if your WebSphere Application Server system is configured to use federated security. |
| ++*DMGRPATH*++ | `/WebSphere/V8ILGDM` | The DManager path in a WebSphere Application Server Network Deployment environment.<br><br>**Important:** After you run HBRUUPTI, check the following data set members to ensure that the DManager path length did not exceed the permitted length and get truncated:<br>► `HBRDSWAS`<br>► `HBRDCWAS`<br>► `HBRDSDVS` |
| ++*WASSERVERNAME*++ | `Serveril2Base` | WebSphere Application Server instance name. |
| ++*PROFILE*++ | `default` | WebSphere Application Server profile. Set to `default` on z/OS. |
| ++*CELLNAME*++ | `cell01` | WebSphere Application Server cell name. |
| ++*NODENAME*++ | `node1` | WebSphere Application Server node name. |
| ++*ADMINHOST*++ | *`washost.ibm.com`* | Name of the host on which the WebSphere Application Server is running. |
| ++*WASBOOTSTRAPPORT*++ | `1234` | Boot strap port that is used by WebSphere Application Server. |
| ++*ADMINUSER*++ | `wasadmin` | Administration console user ID for the WebSphere Application Server administration console. |
| ++*ADMINPSWD*++ | *`t0pS3cr3t`* | Administration console user password for the WebSphere Application Server administration console for the preceding user ID. |
| ++*EJBHLQ*++ | `CLID` | System Authorization Facility (SAF) prefix for EJBROLEs. This might be blank. |

## B.7 WebSphere Optimized Local Adapters script parameters

If you want your COBOL applications to connect to a Rule Execution Server on WebSphere Application Server through WebSphere Optimized Local Adapters (WOLA), customize the variables that are listed in Table B-7. For specific details about WOLA configuration, see Chapter 13, "Configuring IBM WebSphere Optimized Local Adapters support" on page 183.

*Table B-7 Variables for WOLA script parameters*

| Variable | Example value | Description |
|---|---|---|
| ++*HBRWOLALOADLIB*++ | `USER.WOLA.LOADLIB.WAS8` | Load library that is selected as part of setting up WOLA. |
| ++*HBRTARGETRES*++ | `WOLA` | Location for rules execution, in this case, `WOLA`. |
| ++*HBRWOLACELL*++ | `CILK` | Short name of the WebSphere Application Server cell to which to connect using WOLA. |
| ++*HBRWOLANODE*++ | `NILK` | Short name of the WebSphere Application Server node to which to connect using WOLA. |
| ++*HBRWOLASERVER*++ | `WSVR01` | Short name of the WebSphere Application Server for the connection. |

## B.8 WebSphere Application installation script parameters

If you are configuring Operational Decision Manager for z/OS on WebSphere Application Server using **`wsadmin`** scripts, customize the variables that are listed in Table B-8.

*Table B-8 Variables for WebSphere Application installation script parameters*

| Variable | Example value | Description |
|---|---|---|
| ++*RESADMIN*++ | `resAdministrators` | Administrator user group for the Rule Execution Server. |
| ++*RESDEPLOY*++ | `resDeployers` | Deployment user group for the Rule Execution Server. |
| ++*RESMONITOR*++ | `resMonitors` | Monitor user group for the Rule Execution Server. |
| ++*RESADMINUSER*++ | `resAdmin` | Administration user for the Rule Execution Server. |
| ++*RESDEPLOYUSER*++ | `resDeployer` | Deployment user for the Rule Execution Server. |
| ++*RESMONITORUSER*++ | `resMonitor` | Monitor user for the Rule Execution Server. |
| ++*RTSADMIN*++ | `rtsAdministrator` | Administrator user group for Decision Center. |
| ++*RTSCONFIG*++ | `rtsConfigManager` | Configuration user group for Decision Center. |
| ++*RTSUSER*++ | `rtsUser` | User group for Decision Center. |
| ++*RTSINSTALLER*++ | `rtsInstaller` | Installer user group for Decision Center. |
| ++*RTSADMINUSER*++ | `rtsAdmin` | Administration user for Decision Center. |
| ++*RTSCONFIGUSER*++ | `rtsConfig` | Configuration user for the Decision Center. |
| ++*RTSUSERUSER*++ | `rtsUser1` | User for Decision Center. |
| ++*DB2NATIVELOC*++ | /usr/lpp/db2v10/jdbc/lib | The location of the DB2 native library files. |

| Variable | Example value | Description |
|---|---|---|
| *++DB2SERVNAME++* | *myhost.ibm.com* | DB2 host name or IP address. |
| *++DB2PORT++* | 49100 | DB2 connection port. |

## B.9 Subsystem ID used by COBOL management

If you are configuring an execution environment to run COBOL rule subprograms, customize the variables that are listed in Table B-9.

*Table B-9 Subsystem ID used by COBOL management*

| Variable | Example value | Description |
|---|---|---|
| *++R4CSSID++* | `SSID` | Variable to create a new subsystem ID for a COBOL rule subprogram. |

## B.10 WebSphere Application Server Liberty Profile

If you are configuring Operational Decision Manager for z/OS on WebSphere Application Server Liberty Profile, using **`wsadmin`** scripts, customize the variables that are listed in Table B-10.

*Table B-10 Configuration Variables for WebSphere Liberty Profile*

| Variable | Example value | Description |
|---|---|---|
| *++EJBPROF++* | `BBGZDFLT` | The name of the Liberty started task. |
| *++WLPHOME++* | `/usr/lpp/zWebSphere/liberty` | The Liberty profile installation directory. |
| *++WLPSERVER+* | `liberty1` | The name of your Liberty server. |
| *++LIBERTYPORT++* | `20080` | The HTTP port for your Liberty server. |
| *++LIBERTYSECUREPORT++* | `20443` | The HTTP secure port for your Liberty server. |

C

# Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

## Locating the web material

The web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks publication web server:

ftp://www.redbooks.ibm.com/redbooks/SG248014

Alternatively, you can go to the IBM Redbooks website:

http://www.ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248014.

### Downloading the web material

Create a subdirectory (folder) on your workstation, and download the following file of the web material:

```
INSDEMO.cpy
```

# Abbreviations and acronyms

| | |
|---|---|
| **BAL** | Business Action Language |
| **BEP** | Business Event Processing |
| **BOM** | Business object model |
| **BPM** | Business process management |
| **DVS** | Decision Validation Services |
| **HLQ** | High-level qualifier |
| **IBM** | International Business Machines Corporation |
| **ITSO** | International Technical Support Organization |
| **IVP** | Installation verification procedure |
| **JMS** | Java Message Service |
| **JVM** | Java virtual machine |
| **KPI** | Key performance indicator |
| **RES** | Rule Execution Server |
| **SSP** | Scenario service provider |
| **XU** | Execution unit |
| **zFS** | z/OS Distributed File Service |
| **zRES** | zRule Execution Server for z/OS |

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that publications referenced in this list might be available in softcopy only.

- *Implementing Event Processing with CICS*, SG24-7792
- *Proven Practices for Enhancing Performance: A Q & A for IBM WebSphere ILOG BRMS 7.1*, REDP-4775
- *Making Better Decisions Using IBM WebSphere Operational Decision Management*, REDP-4836
- *Patterns: Integrating WebSphere ILOG JRules with IBM Software*, SG24-7881

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

**ibm.com**/redbooks

## Other publications

These publications are also relevant as further information sources:

- Barbara von Halle, *Business Rules Applied*, Wiley, 2001, ISBN 978-0471412939
- Steve Craggs & Brian Safron, *Operational Decision Manager for Dummies*, Wiley, 2013, ISBN 978-1118679784

  http://www.ibm.com/software/de/beweglich-bleiben/pdf/IBM_ODM_for_Dummies.pdf

## Online resources

These websites are also relevant as further information sources:

- IBM Operational Decision Manager product website:

  http://www.ibm.com/software/decision-management/operational-decision-management/websphere-operational-decision-management
- IBM WebSphere Operational Decision Management Version 8.6.0 IBM Knowledge Center:

  http://www.ibm.com/support/knowledgecenter/SSQP76_8.6.0
- Video: Optimizing Decision Management with IBM WebSphere and System z

  https://www.youtube.com/watch?v=zc96l33NP_g

- Video: IBM Operational Decision Manager V8.5 demo

  https://www.youtube.com/watch?v=NCPz-Lxicio

- WebSphere z/OS Optimized Local Adapters
  http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101490
- The ilog.rules.dvs.client package extensive online documentation with code samples that explain how to use it:

  http://www.ibm.com/support/knowledgecenter/SSQP76_8.7.1/com.ibm.odm.dserver.rules.ref.designer/html/api/html/ilog/rules/dvs/client/package-summary.html
- *Decision Management: Enabling Faster, More Consistent Business Decisions in Enterprise Applications*

  ftp://public.dhe.ibm.com/software/systemz/pdf/System_z_eBook_Decision_Management.Final.4.1.14.pdf
- CICS Explorer:

  http://www.ibm.com/software/htp/cics/explorer

## Help from IBM

IBM Support and downloads

**ibm.com**/support

IBM Global Technology Services

**ibm.com**/services

Redbooks

Flexible Decision Management with Business Rules on IBM z Systems

SG24-8014-02

ISBN 0738440728

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8014-02

ISBN 0738440728

Printed in U.S.A.