IBM® WebSphere®

IBM Operational Decision Management V8.0 Performance Tuning Guide

Design choices for events and business rule applications

Runtime tuning for events and rule applications

Tuning for Decision Center

David Granshaw
Daniel McGinnes
Pierre-Andre Paumelle

Redpaper

ibm.com/redbooks

International Technical Support Organization

IBM Operational Decision Management V8.0 Performance Tuning Guide

January 2013

REDP-4899-00

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

First Edition (January 2013)

This edition applies to IBM WebSphere Operational Decision Management Version 8.0 and IBM Operational Decision Manager V8.0.1.

This document was created or updated on January 29, 2013.

Contents

Notices

Trademarks

Preface

IBM® Operational Decision Management (ODM) is a family of products used by IT and business users to create and manage business decision logic throughout their organization. This IBM Redpaper™ publication offers advice on all aspects of performance, including hardware, architecture, authoring, quality of service, monitoring, and tuning. The advice is based upon preferred practices and experience gained from real customer situations.

This paper is aimed at a wide ODM audience, including IBM employees and customers, and provides useful information to both new and experienced users.

Although the product family is known as IBM WebSphere® Operational Decision Management (WODM), at V8.0, with V8.0.1 the the name is now simply IBM Operational Decision Manager (ODM). The performance information in this paper is based on V8.0 of this product family and differences introduced with V8.0.1 are pointed out.

The team who wrote this paper

David Granshaw is a Senior Engineer and WODM performance architect at the IBM Lab in Hursley, UK. He previously led the WebSphere Application Server default messaging performance team and has worked on performance across a number of products in the WebSphere portfolio. Before joining IBM, David was a Senior Scientist, working for ten years in government scientific research.

Daniel McGinnes is a Performance Specialist at the IBM Lab in Hursley, UK. He currently works on measuring and tuning performance of Operational Decision Management. Daniel previously worked on the performance of other WebSphere portfolio products, including WebSphere Business Events, WebSphere Application Server and WebSphere Enterprise Service Bus. He has also worked as an IT specialist providing consulting services to IBM customers across EMEA and USA.

Pierre-Andre Paumelle is a Performance Architect in Operational Decision Management with over 15 years of experience designing and developing enterprise applications with C++ and Java (Java EE). He worked with clients for eight years as a consultant on IBM ILOG® products, especially those products in the BRMS product line. Pierre-Andre has worked for five years as the architect and team lead on the Enterprise Integration (BRMS) team responsible for Rules Execution Server, Decision Warehouse, and Decision Validation Services.

A special thank you to Dan Selman for his contribution to the Rule Designer and Rule engine chapters.

Daniel Selman is a Software Architect and a team lead for IBM WebSphere ILOG JRules Rule Studio, which is a rule development environment for technical users that is based on the Eclipse platform. He has worked as an Architect and Product Manager on a variety of Java C, C++, and Java EE applications. Daniel blogs at:

https://www.ibm.com/developerworks/mydeveloperworks/blogs/brms/?lang=en

Thanks to the following people for their contributions to this project:

Carla Sadtler, Deana Coble, Linda Robinson
International Technical Support Organization, Raleigh Center

Tim Galvin, IBM Littleton

Lewis Evans, IBM Hursley

Now you can become a published author, too!

Comments welcome

Stay connected to IBM Redbooks

- 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`

1 Overview

This chapter introduces IBM Operational Decision Management (ODM) and discusses the overall structure of the book.

This chapter includes the following topics:

- IBM Operational Decision Management
- Publication structure

1.1 IBM Operational Decision Management

IBM Operational Decision Management (ODM) is a family of products that are used by IT and business users to create and manage business decision logic that is used throughout their organizations. It enables businesses to respond to real-time data with intelligent, automated decisions.

Operational Decision Management consists of two key components:

- WebSphere Decision Center provides an integrated repository and enables management of business rule and event-based logic. It provides comprehensive decision governance, allowing IT and business users to work collaboratively across an entire organization.
- WebSphere Decision Server provides Eclipse-based tooling and high performance run times, Decision Server Rules, and Decision Server Events, for rule and event processing.

This book offers advice on all aspects of performance, including hardware, architecture, authoring, quality of service, monitoring, and tuning. The advice is based upon preferred practices and experience gained from real customer situations.

1.2 Publication structure

The following list summarizes each chapter of this document:

- Chapter 1, “Overview” on page 1

  The current chapter presents an overview of the book.
- Chapter 2, “General performance tuning” on page 5

  This chapter discusses some general performance tuning advice that is applicable to Decision Server Rules, Decision Server Events, and Decision Center. The key topics are hardware, monitoring, and JVM tuning.
- Chapter 3, “Decision Server Events: Designing for performance” on page 11

  This chapter discusses the options available when writing projects with Event Designer and demonstrates the performance impact of those choices. The key topics are durability, simple versus complex event rules, rule ordering, and complex scoped business objects.
- Chapter 4, “Decision Server Events: Run time performance tuning” on page 25

  This chapter discusses the tuning of the Decision Server Events run time. It includes the tuning of threads, databases, messaging providers, and technology connectors. It also covers cluster topologies, history, tracing, monitoring, testing, and diagnosis of performance issues.
- Chapter 5, “Decision Server Rules: Rule Designer tuning” on page 61

  This chapter discusses the options available when writing projects with Rule Designer. The key topics are build tuning, synchronization, benchmarking, and analysis of rule characteristics.
- Chapter 6, “Decision Server Rules: Rule engine execution tuning” on page 65

  This chapter discusses the run time tuning of the rule execution engine. The key topics are academic benchmarks, execution modes, and tips and tricks for business rule applications.

- Chapter 7, “Decision Server Rules: Rule Execution Server tuning” on page 75

  This chapter discusses the run time tuning of the rule execution server. The chapter covers the performance impact of the executable object model (XOM), trace, Decision Warehouse, topology and the rule processing algorithm. It also covers architecture choices, tuning, and benchmarking.
- Chapter 8, “Decision Server Rules: Tuning Decision Warehouse” on page 91

  Decision Warehouse traces decisions in production applications, storing results of rule sets executed outside of Decision Validation Services. This chapter discusses the features, tuning, and performance impact of Decision Warehouse.
- Chapter 9, “Decision Server Rules: Decision Validation Services tuning” on page 97

  Decision Validation Services is used to test and validate rule scenarios. This chapter discusses recommendations and tuning.
- Chapter 10, “Decision Center tuning” on page 101

  WebSphere Decision Center is an integrated repository and management of business rule and event-based logic. This chapter discusses the tuning of Decision Center.

2

General performance tuning

This chapter discusses general performance tuning for IBM Operational Decision Management Version 8.0 software. Operational decision management is business rules software that simplifies and automates the process of authoring, changing, and executing operational decisions. The concepts covered in this chapter are applicable to Decision Server Rules, Decision Server Events, and Decision Center. These components work together as a part of IBM Operational Decision Management software. In combination, these components streamline operational decisions by enabling users to define, change, and test business rules and events.

This chapter covers the following topics:

- Hardware and monitoring
- Tuning the Java virtual machine (JVM)

2.1 Hardware and monitoring

The choice of hardware is one of the most critical factors affecting performance and it is important to ensure that the hardware is correctly sized for the expected workload.

Once the hardware has been chosen, it is important to monitor the cpu utilization, disk IO, network IO, memory, and paging. Monitoring is conducted while the system is running a representative workload to identify any performance bottlenecks. These statistics can be measured with operating-system-specific tools such as **`perfmon`** for Windows, or **`iostat`** or **`vmstat`** for Linux and UNIX. Understanding the performance bottlenecks allows tuning to be targeted where it provides the greatest benefit.

2.2 Tuning the Java virtual machine (JVM)

JVM tuning is important because Decision Server Rules and Decision Server Events are both Java Platform, Enterprise Edition (Java EE) applications that run on WebSphere Application Server. Optimal performance is therefore dependent upon the correct tuning of Decision Server, WebSphere Application Server, and the underlying JVM. Two key aspects of JVM tuning are the garbage collection policy and the JVM heap size. The JVM heap is an area of memory where objects used in a Java program are allocated.

2.2.1 Garbage collection policies

When the Java heap is insufficient and unable to satisfy a request for storage, such as object creation, the underlying JVM automatically triggers garbage collection to free memory. The key garbage collection policy choices are considered in the following list:

- Optthruput Garbage Collection (default prior to WebSphere Application Server V8)

  The Optthruput policy uses a single Java heap and will consume the entire heap before pausing the application and invoking a garbage collection.

- Optavgpause Garbage Collection

  The Optavgpause policy also uses a single heap but performs as much garbage collection processing as possible in parallel with application processing. This reduces the time that the application is paused during garbage collection.

- Generational Garbage Collection (default in WebSphere Application Server V8)

  The Generational policy uses two Java heaps, one for short-lived objects (nursery heap) and one for longer-lived objects (tenured heap). The two heaps can garbage collect independently, which improves overall efficiency. Relative to Optthruput, efficiency is improved typically by 5% for Operational Decision Management workloads.

> **Note:** Generational Garbage Collection is the default for WebSphere Application Server V8 and is the preferred choice for most Operational Decision Management applications.

- Balanced Garbage Collection (new in WebSphere Application Server V8)

  The Balanced policy is new in WebSphere Application Server V8 and splits the Java heap into regions that can be collected independently. It is designed to reduce the occasional long pause times that can be associated with large heaps. However, the Balanced policy produces a slight performance degradation relative to generational garbage collection. It is designed for 64-bit WebSphere Application Servers with heaps greater than 4 GB.

2.2.2 Java heap size

If the Java heap is too small, then garbage collection will occur too frequently, consuming a significant number of cycles reanalyzing the heap. Conversely, if the heap is too large, long pause times will occur each time the heap is analyzed and garbage collected (unless you are using the Balanced Garbage Collection policy).

Care must be taken to ensure that the chosen heap size does not cause paging. Paging can occur if there is insufficient RAM to support the heap size in addition to the other processes running on the system. As a result, performance will be significantly degraded because data must to be continually written and read from disk rather than main memory. Paging can be monitored using system-specific tools such as **`perfmon`** (Window servers) or **`vmstat`** (Linux, UNIX). If significant paging does occur, consider reducing the heap size, adding more system memory, and shutting down any unnecessary processes running on the system. If Decision Server Events is using a local repository database, consider moving the database to a remote system to reduce memory contention.

2.2.3 JVM preferred settings

The default, Generational garbage collection policy, is the preferred policy for most Operational Decision Management applications and is tuned with three key properties. Those properties are minimum heap size, maximum heap size, and nursery size (the nursery must be smaller than the minimum heap size). The optimal configuration will depend upon the nature of the workload and the available memory. Table 2-1 shows values that are generally suitable for a range of ODM workloads.

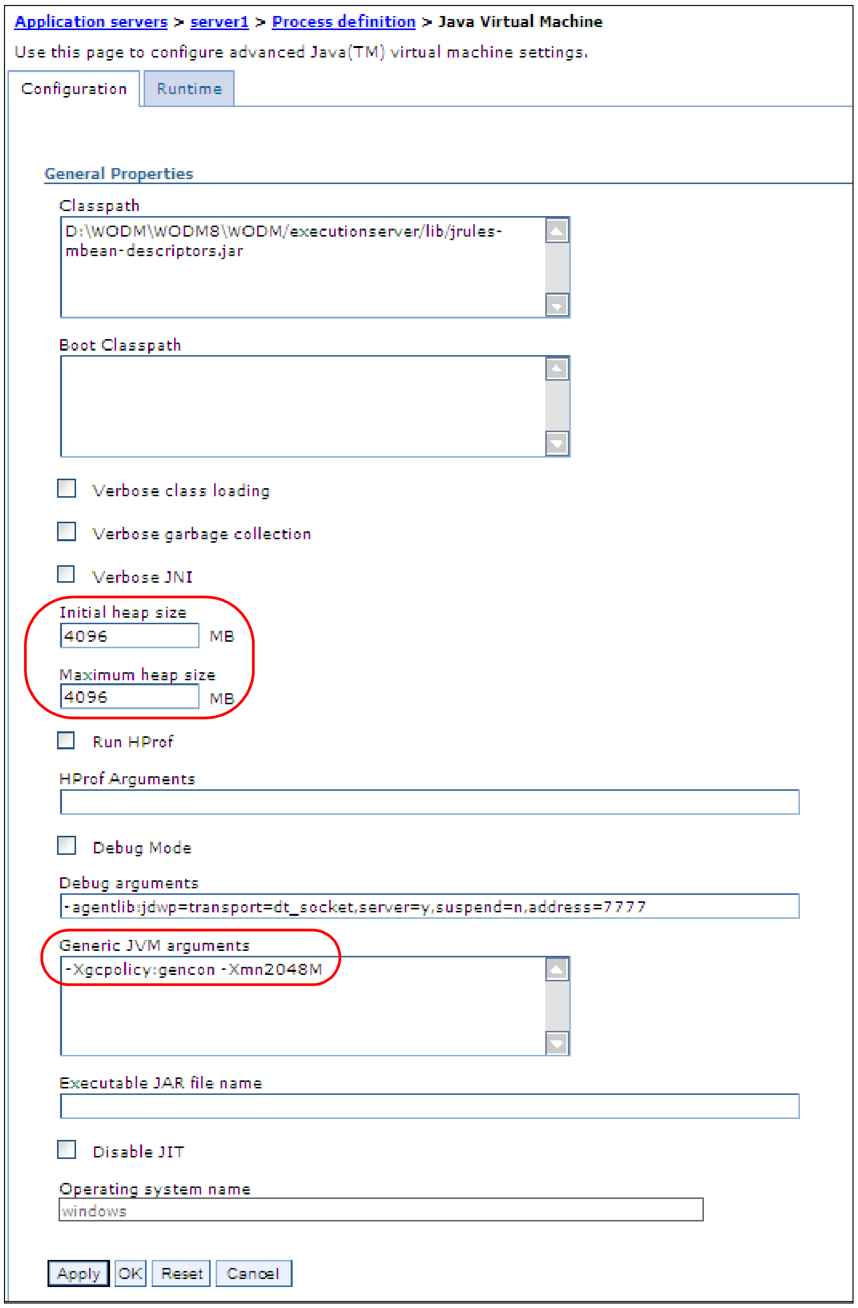
Table 2-1 JVM settings

	32-bit JVM with 32 GB User Address Space limit (for example, Windows 32 or zLinux 31)	**64-bit JVM**
Initial heap size	1280 MB	4096 MB
Maximum heap size	1280 MB	4096 MB
Generic JVM arguments	-Xgcpolicy:gencon -Xmn1024M	-Xgcpolicy:gencon -Xmn2048M

The first two parameters set the initial and maximum heap size. The third parameter sets the generational garbage collection policy and the size of the nursery heap. The nursery is used for short-lived objects; the remainder of the heap will be used for longer-lived objects.

The parameters can be set using the WebSphere Application Server administration console. To set them, select **Application Servers** → ***server 1*** → **Java and Process Management** → **Process Definition** → **Java Virtual Machine.**

Figure 2-1 shows how these properties are set in the WebSphere Application Server console for a 64-bit JVM.


Application servers > server1 > Process definition > Java Virtual Machine

Use this page to configure advanced Java(TM) virtual machine settings.

Classpath: D:\WODM\WODM8\WODM/executionserver/lib/jrules-mbean-descriptors.jar

Boot Classpath:

[ ] Verbose class loading

[ ] Verbose garbage collection

[ ] Verbose JNI

Initial heap size: 4096 MB

Maximum heap size: 4096 MB

[ ] Run HProf

HProf Arguments:

[ ] Debug Mode

Debug arguments: -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=7777

Generic JVM arguments: -Xgcpolicy:gencon -Xmn2048M

Executable JAR file name:

[ ] Disable JIT

Operating system name: windows

Figure 2-1 General JVM tuning on a 64-bit operating system

Additional information about IBM JVM tuning, together with advice on Solaris and HP-UX JVMs, can be found in the WebSphere Application Server Information Center:

`http://www14.software.ibm.com/webapp/wsbroker/redirect?version=compass&product=was-express-dist&topic=container_tune_jvm`

If IBM Operational Decision Management is running on IBM z/OS®, consider tuning both the adjunct and servant JVMs.

If the warmup of a Rule Execution Server is taking a long time, consider the following hints when configuring the JIT:

- The JVM supplied with WebSphere Application Server version 8.0.0.3 generates more JIT code than previous versions. More JIT code can lead to longer warm-up times and lower throughput.
- The JIT can take time to compile a large rule set, but this behavior can be modified by increasing the JIT cache or by limiting the inlining on ODM classes. The following list shows both options:
  - Cache size and number compilation threads:

    ```
    -Xjit:codeTotal=261072 compilationThreads=1
    ```

  - Limit the inlining for one of the ODM packages:

    ```
    -Xjit:{ilog/rules/engine/sequential/generated/*}(disableInlining)
    ```

3

Decision Server Events: Designing for performance

Decision Server Events is the business event processing module of Decision Server. This module enables businesses to detect, evaluate, and respond to event patterns. This chapter discusses the options available when writing projects with Event Designer and demonstrates the performance impact of those choices.

This chapter covers the following topics:

- High-level considerations for optimal design
- Optimizing throughput by minimizing disk access
- Performance impact of durability
- Relative performance of simple and complex event rules
- Event rule ordering
- Context scoped business objects
- Multiple event rules and synthetic events
- Impact of the technology connectors on performance

3.1 High-level considerations for optimal design

A number of design choices affect the performance of a Decision Server Events system. This section provides suggestions for optimal design. Each of the concepts identified here is covered in more detail later in this chapter. The following list outlines options and resources for your consideration during the design process:

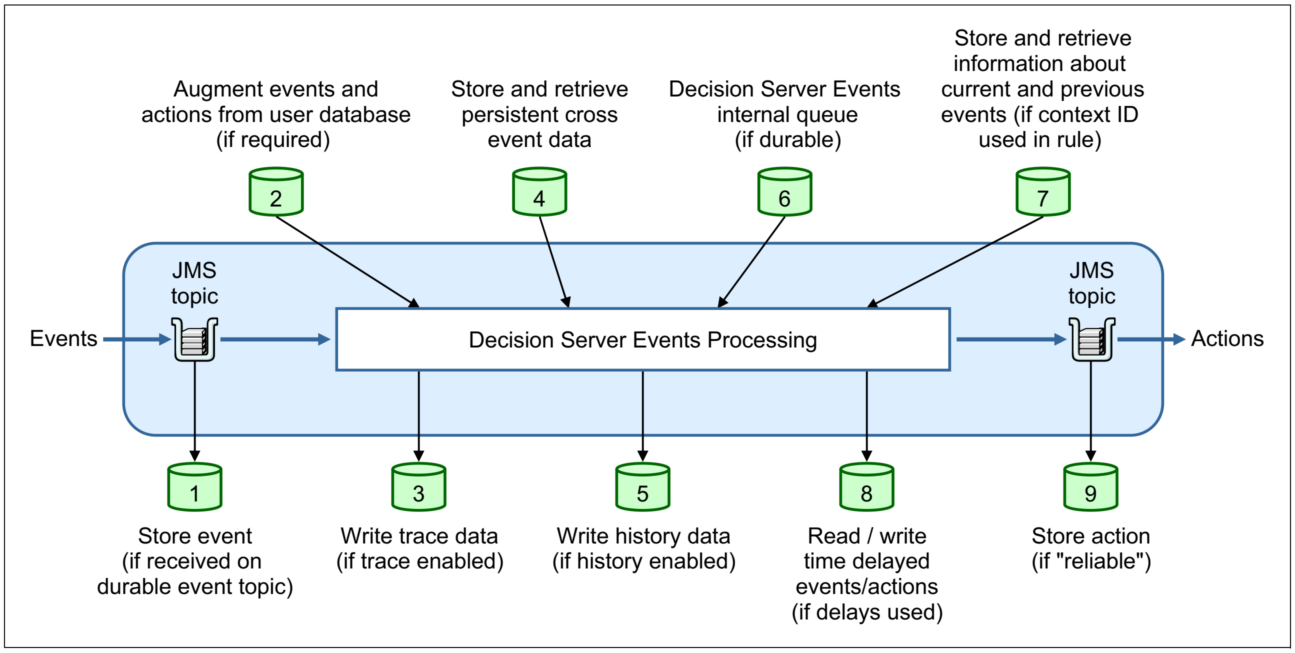
- Ensure that you understand how your design will affect disk usage and seek to minimize disk access where possible. See 3.2, "Optimizing throughput by minimizing disk access" on page 12.
- Ensure that you are using appropriate durability for Events and Actions. See 3.3, "Performance impact of durability" on page 15.
- When designing your project, ensure that context IDs and context scoped business objects are used appropriately; order your filters for optimal performance. See 3.4, "Relative performance of simple and complex event rules" on page 16 and 3.5, "Event rule ordering" on page 17.
- When using non-persistent context scoped business objects, consider using generational garbage collection with a tenured heap that is large enough to contain all the context scoped business object (CSBO) data. See 3.6.1, "Memory usage" on page 20.
- Where possible, send events in the native Decision Server format directly to the default JMS event destination. This avoids the need to use the event connectors for protocol conversion and message transformation. Similarly, consider receiving actions in the native Decision Server Events format directly from the default JMS action topic. See 3.8, "Impact of the technology connectors on performance" on page 23.

3.2 Optimizing throughput by minimizing disk access

At several points during processing, Decision Server Events persists data to disk or reads data from disk. This is required for a number of reasons, such as, the provision of durable processing, recording history, logging or tracing, and the storage and retrieval of runtime data. Minimizing disk access significantly improves performance. The following section discusses the reasons why disk access can occur. For optimal performance, ensure disks are only accessed when strictly necessary.

3.2.1 Decision Server disk access points

Figure 3-1 on page 13 shows the reasons why Decision Server Events requires disk access (the numbered disk access points are used in the following discussion).



Augment events and actions from user database (if required)
Store and retrieve persistent cross event data
Decision Server Events internal queue (if durable)
Store and retrieve information about current and previous events (if context ID used in rule)
2
4
6
7
JMS topic
JMS topic
Events
Decision Server Events Processing
Actions
1
3
5
8
9
Store event (if received on durable event topic)
Write trace data (if trace enabled)
Write history data (if history enabled)
Read / write time delayed events/actions (if delays used)
Store action (if "reliable")

Figure 3-1 Decision Server Events disk access points

- **JMS provider (numbers 1 and 9 in Figure 3-1) - persisting events and actions**

  All events and actions are placed on a JMS queue or topic immediately before event processing and after an action has been fired. This is also true when technology connectors are used. The one exception is events that are delivered by a WebSphere eXtreme Scale client, which delivers events directly to an internal, non-JMS queue. The WebSphere eXtreme Scale client is outside the scope of this document.

  The JMS provider writes events and actions to disk if processing is durable or if a large number of non-durable events or actions are currently in memory. The durability of events has a significant impact on performance, as discussed in 3.3, “Performance impact of durability” on page 15.

- **Logging and tracing (number 3 in Figure 3-1)**

  Logging and tracing have a significant impact on performance and should be minimized or eliminated wherever possible. The performance implications of enabling trace are covered in 4.4, “The impact of logging and tracing” on page 31.

- **User-defined databases and files (number 2 in Figure 3-1)**

  Increased disk utilization will occur if events and actions require access to a local user database, for example, database fetches in a business object for event and action augmentation. If you are using a local database and experiencing disk contention or the Decision Server Events server is close to 100% CPU utilization, consider moving these databases to a remote machine with fast disk drives.

- **History (number 5 in Figure 3-1)**

  When history is enabled, a significant amount of information about events, actions, and filters is written to the Decision Server Events repository. Unless strictly required, history should be disabled. The performance implications of enabling history are covered in 4.3, “The impact of history on performance” on page 28.

- **Defining a context ID on event rules (number 7 in Figure 3-1)**

  When event rules and event rule groups have a context relationship defined, each processed event must write a small amount of context information to the Decision Server

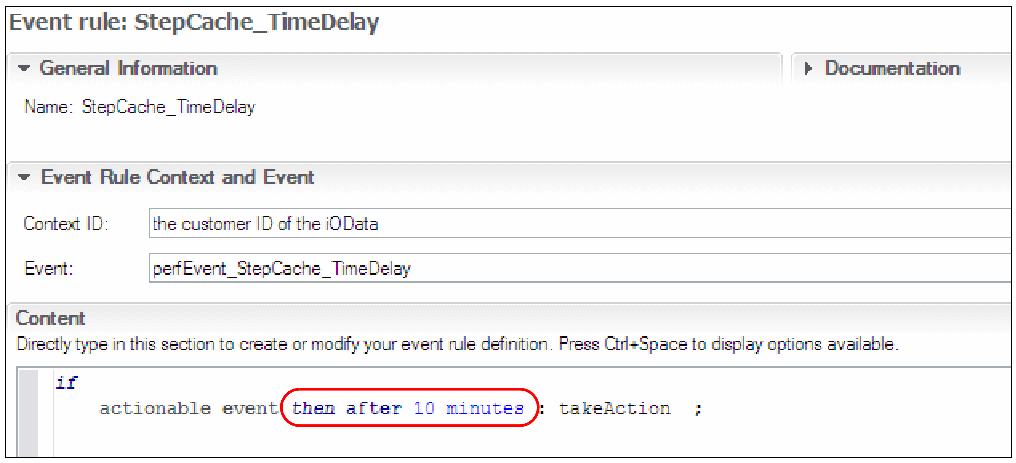
Events repository database. This context information is retrieved during rule processing to determine the temporal relationship between events. To reduce disk utilization and improve performance, define context relationships only if absolutely necessary, such as when the processing of the events has a dependency on previous or future events. The performance impact of using context relationships is covered in 3.4, “Relative performance of simple and complex event rules” on page 16.

- **Context scoped business objects (number 4 in Figure 3-1)**

  Context scoped business objects (CSBO) can be used to share business data between multiple events in the same context. Examples of CSBOs are holding an average balance (CSBO summary object) or keeping track of the last three cash withdrawals for each customer (CSBO array object). The number of entries in a CSBO array can be restricted by time or by explicitly setting a limit on the number of entries allowed in the array. The performance impact of using CSBOs is covered in 3.6, “Context scoped business objects” on page 19.

- **Time-delayed events and actions (number 8 in Figure 3-1)**

  Events and actions can be either executed immediately or delayed and executed later. When time delays are used, they cause events and actions to be written to the Decision Server Events repository database, where they are held until the required execution time. Adding time delays therefore increases database and disk usage. Figure 3-2 shows an example where actions are delayed by 10 minutes. If you are using a local repository database and experiencing disk contention, or close to 100% CPU utilization, consider moving the repository to a remote server with fast disk drives.



Name: StepCache_TimeDelay

Context ID: the customer ID of the iOData

Event: perfEvent_StepCache_TimeDelay

Directly type in this section to create or modify your event rule definition. Press Ctrl+Space to display options available.

```
if
    actionable event then after 10 minutes : takeAction  ;
```

Figure 3-2 Time Delays in an Event Rule

- **Decision Server Events internal queue (number 6 in Figure 3-1 on page 13)**

  If events are sent directly to the Decision Server durable event destination or if event connectors are set to reliable, event data is written to disk. Event data is queued to be written between the preprocessing (xml parsing and Context ID evaluation) and final rule processing. This is an internal queue and ensures durability in the event of a hardware or software failure. If you are experiencing disk contention, consider installing faster disk drives (RAID/SAN) for Decision Server Events.

3.2.2 Additional disk access points

Technology connectors are internal system components that provide codeless connection to and from touchpoints by using a variety of protocols. The following technology connectors can generate additional disk access:

- Message queue connectors (if events or actions are persistent)
- RDBMS connector (due to database activity)
- File system connector (due to file system activity)

Preferred practice: Minimizing all types of disk access

- Use non-persistent, non-durable event processing where practical.
- Use logging only when required.
- Disable history unless required.
- Use CSBO, time-delays, and Context IDs only when necessary.

3.3 Performance impact of durability

Decision Server Events supports non-durable and durable processing. The two types of processing are described as follows:

- Non-durable processing means events are sent to the Decision Server Events `eventDestination` with a JMS delivery mode of non-persistent. Non-durable events and actions are stored in memory unless the allocated memory buffers become full. Non-durable events are not recovered if Decision Server Events or the messaging provider restarts.
- Durable processing means events are sent to the Decision Server Events `durableEventDestination` with a JMS delivery mode of persistent. Durable events and actions are persisted to disk and will be recovered if Decision Server Events or the underlying messaging provider restarts.

  If you send events to the `durableEventDestination` with a JMS delivery mode of non-persistent, events could still be lost if the server restarts.

It is up to the system administrator to balance system resources and recovery needs.

You should also ensure that actions are marked with the appropriate subscription type on the Connector tab of the action in Event Designer. This setting controls whether actions are sent with a JMS delivery mode of persistent to the `durableActionTopic` (Reliable), or JMS delivery mode of non-persistent to the `actionTopic` (Express).

For information about configuring durable and non-durable events and actions, refer to the following website:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dserver.events.dev/topics/tsk_config_dur_nondur_events_actions.html

Carefully consider the durability requirements for your event processing because this has a large impact on Decision Server Events performance.

As an example, we ran a test scenario using complex event rules, javascript, synthetic events, and database enrichment, with one action being generated for every 100 events. In our tests, we obtained results showing that:

- Non-durable processing was 2.4 times faster than durable processing when using WebSphere Application Server default messaging as the JMS provider.

- Non-durable processing was 2.5 times faster than durable processing when using remote WebSphere MQ as the JMS provider.

Preferred practice: Durability

Carefully consider your durability requirements and use non-durable processing where possible.

3.4 Relative performance of simple and complex event rules

Event rules define the logic that will be evaluated in response to events. This section shows how adding a context ID to event rules can affect performance.

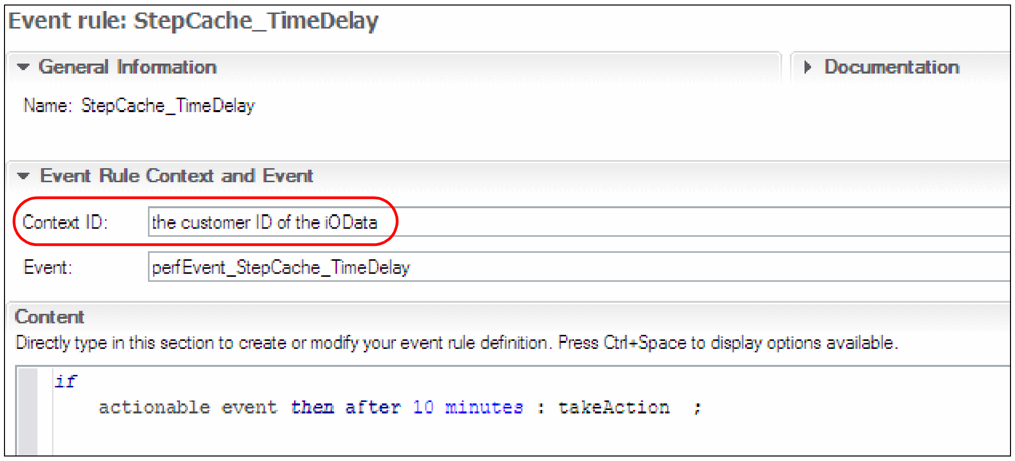
3.4.1 Simple event rules

Using simple event rules wherever possible significantly improves performance. By definition, a simple event rule can be evaluated without reference to previous events, for example, `is the value of this order greater than £200`. This is the simplest form of event rule processing and produces the highest throughput.

3.4.2 Simple event rules with a context ID

If a context ID is added to a simple event rule, as shown in Figure 3-3, this indicates to Decision Server Events that information about the occurrence of the event will be used in the evaluation of future complex event rules. As a result, information about the occurrence of the event is stored in the Decision Server Events repository database, which adds path length and increases database activity. This results in lower throughput than a simple event rule without a context ID.

V8.0.1 update: From V8.0.1 onwards, the addition of a context ID to a simple rule will not automatically result in information about the occurrence of an event being written to the Decision Server Events repository. Information will be written only if it is referenced by a complex event rule.



Name: StepCache_TimeDelay

Context ID: the customer ID of the iOData

Event: perfEvent_StepCache_TimeDelay

Directly type in this section to create or modify your event rule definition. Press Ctrl+Space to display options available.

```
if
    actionable event then after 10 minutes : takeAction  ;
```

Figure 3-3 Specifying a Context ID

3.4.3 Complex event rules

By definition, complex event rules have a dependency on previous events and will therefore always have a context ID defined. The context ID (for example, a customer reference number) is used to determine which events have a relationship. The following lines are examples of a complex event rule for a particular customer or context ID:

```
if event A has occurred twice before
if event A occurred before event B
```

When complex event rules are processed, information about the current event must be stored in the Decision Server Events repository database, and information about previous events with the same context ID must be retrieved from the database. The increased processing and database access reduces performance as compared to simple event rules.

The difference in path length and database activity makes a significant difference in throughput. In our test scenario using non-durable, non-persistent processing on a single server, we obtained results showing that:

- The throughput using simple event rules without a context ID can be up to a factor of 2 higher than the throughput for complex event rules.
- The throughput using simple event rules without a context ID can be 20% higher than the throughput for simple event rules with a context ID.

For the same test scenario using durable, persistent processing, we obtained results showing:

- The throughput using simple event rules without a context ID can be 20% higher than the throughput for complex event rules.
- The throughput using simple event rules without a context ID can be 10% higher than the throughput for simple event rules with a context ID.

Simple event rules without a context ID also have significantly better performance in a Decision Server Events cluster. Consider this option a useful way of filtering out some events before performing complex event processing.

V8.0.1 update: From V8.0.1 onwards using a single server, the addition of a context ID to a simple event rule will not have a significant performance impact if the information about the associated events and actions is not referenced in a complex event rule.

In a cluster configuration, the addition of a context ID to a simple event rule will still have a significant overhead because events might need to be serialized between cluster members.

Preferred practice: Use of context IDs

Define a context ID for event rules only when required.

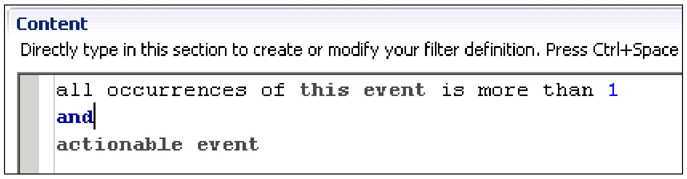
3.5 Event rule ordering

The order in which event rules are defined can also have a significant effect on performance. Consider the following filters:

- Simple filter (actionable event): Checks the value of an event field.

- Complex filter (all occurrence of this event is more than 1): Requires information about previous events to be retrieved from the Decision Server Events repository database. This requires more resources to evaluate than the simple filter.

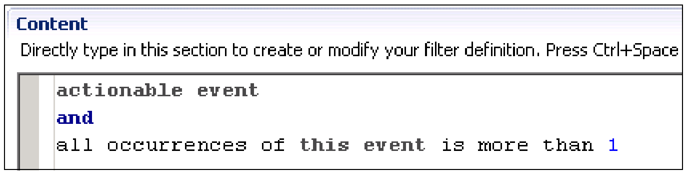
These filters are shown in Figure 3-4 and Figure 3-5. In Figure 3-4, the complex filter is evaluated first, followed by the simple filter.


Content
Directly type in this section to create or modify your filter definition. Press Ctrl+Space

```
all occurrences of this event is more than 1
and
actionable event
```

Figure 3-4 Filter A: Complex filter first

In Figure 3-5, the order of the filters is reversed, so the simple filter is evaluated first.


Content
Directly type in this section to create or modify your filter definition. Press Ctrl+Space

```
actionable event
and
all occurrences of this event is more than 1
```

Figure 3-5 Filter B: Simple filter first

Functionally, these two event rules will behave identically. However, their performance can be quite different. Decision Server Events event rules are always evaluated from top to bottom. Because this is an *and* operation, if the first filter evaluates to false, there is no need to execute the second filter. Putting the simple filter first increases processing speed because the complex filter is only evaluated when the simple filter is returned as true.

As an example, we ran a test scenario with these event rules in which the *actionable event* returned true for one in every 100 events. The more expensive complex filter was therefore only evaluated one in every 100 events. In our tests, we obtained results showing:

- For non-durable processing, placing the simple filter first produced a 50% performance gain.
- For durable processing, placing the simple filter first produced a 20% performance gain.

If the actionable event returned true for a greater proportion of events, then this improvement would be reduced. If it returned true for every event, then the performance for both event rules would be the same.

This principle can be applied to other types of filters, for example, to those using a field populated by database enrichment or JavaScript. The database lookup or JavaScript is only executed when the intermediate object field is referenced, so the most simplistic and fastest filters should be placed first in event rules.

Preferred practice: Rule ordering

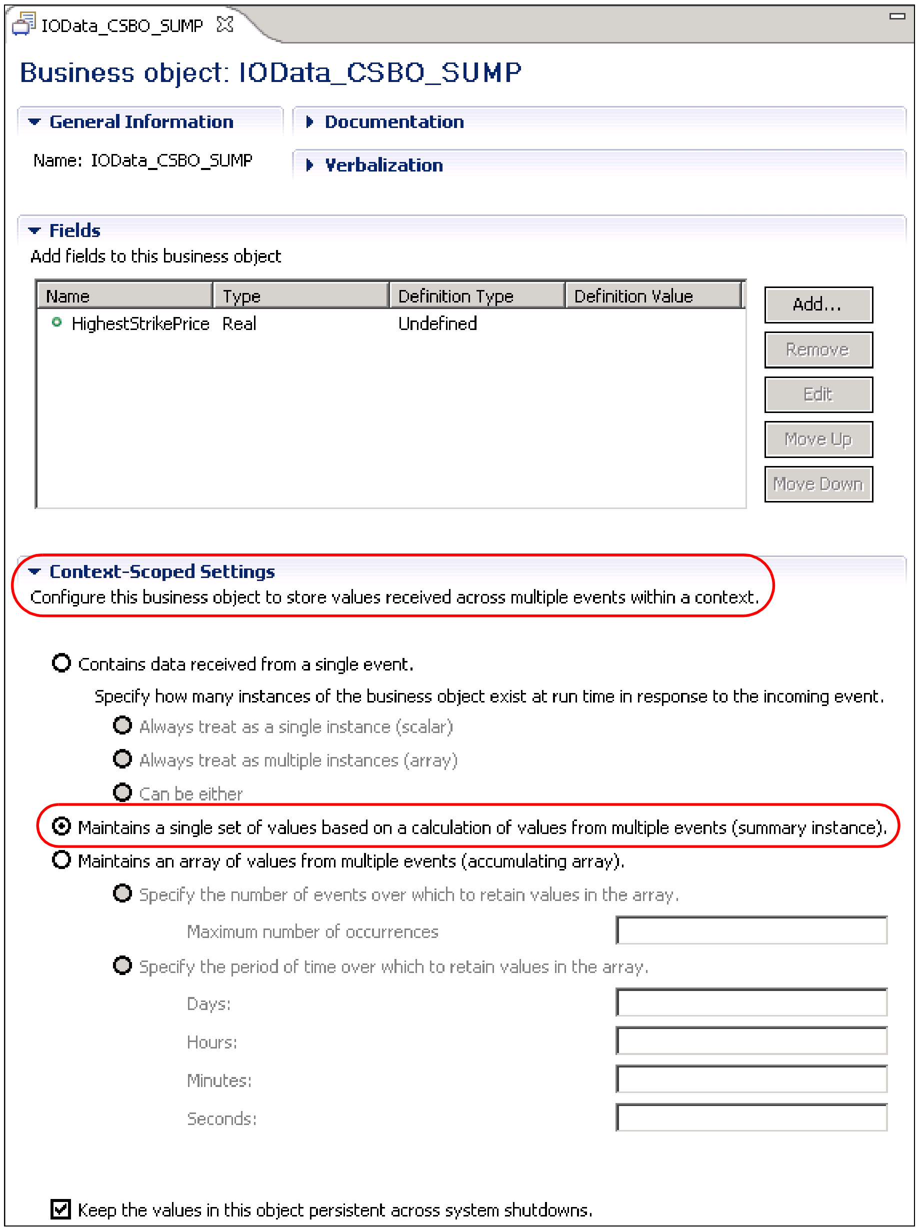
Analyze your event flows and order your rule sets so simple filters, especially if they often evaluate to false, are executed ahead of more complex ones.

3.6 Context scoped business objects

Information can be shared across events in a context using context scoped business objects (CSBOs). The following list shows the two types of CSBOs:

- A summary instance CSBO. In this case, one CSBO is stored for each context. For example, a bank might want to keep a running average of cash withdrawals for each customer.

  CSBO summary instances can be set using the context scoped settings on the business object, as shown in Figure 3-6.



Business object: IOData_CSBO_SUMP

Name: IOData_CSBO_SUMP

Fields

Add fields to this business object

Name	Type	Definition Type	Definition Value
HighestStrikePrice	Real	Undefined	

Context-Scoped Settings

Configure this business object to store values received across multiple events within a context.

( ) Contains data received from a single event.

Specify how many instances of the business object exist at run time in response to the incoming event.

( ) Always treat as a single instance (scalar)

( ) Always treat as multiple instances (array)

( ) Can be either

[✓ Maintains a single set of values based on a calculation of values from multiple events (summary instance).]

( ) Maintains an array of values from multiple events (accumulating array).

( ) Specify the number of events over which to retain values in the array.

Maximum number of occurrences

( ) Specify the period of time over which to retain values in the array.

Days:

Hours:

Minutes:

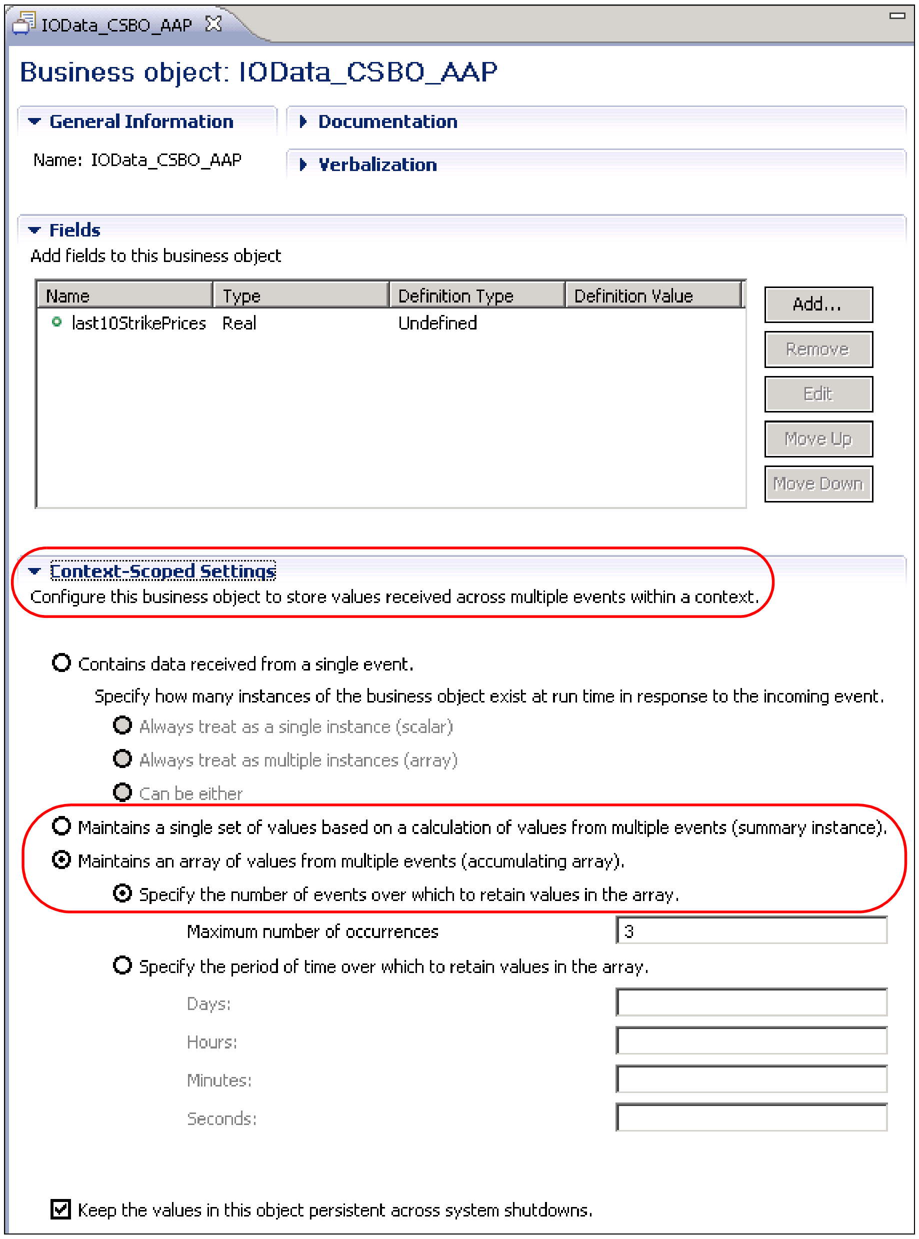
Seconds:

[✓] Keep the values in this object persistent across system shutdowns.

Figure 3-6 Configuring a context scoped business object summary instance

- A CSBO array. In this case, an array of CSBOs are stored for each context. The number of array entries can be limited by number or by time. For example, a bank might want to record the last five cash withdrawals that each customer has made or record each customer's withdrawals in the last 24 hours.

CSBO arrays can be set using the context scoped settings on the business object, as shown in Figure 3-7.


IOData_CSBO_AAP

Business object: IOData_CSBO_AAP

General Information

Name: IOData_CSBO_AAP

Documentation

Verbalization

Fields

Add fields to this business object

Name	Type	Definition Type	Definition Value
last10StrikePrices	Real	Undefined	

Context-Scoped Settings

Configure this business object to store values received across multiple events within a context.

- [ ] Contains data received from a single event.

  Specify how many instances of the business object exist at run time in response to the incoming event.

  - [ ] Always treat as a single instance (scalar)
  - [ ] Always treat as multiple instances (array)
  - [ ] Can be either
- [ ] Maintains a single set of values based on a calculation of values from multiple events (summary instance).
- [✓] Maintains an array of values from multiple events (accumulating array).
  - [✓] Specify the number of events over which to retain values in the array.

    Maximum number of occurrences: 3
  - [ ] Specify the period of time over which to retain values in the array.

    Days:

    Hours:

    Minutes:

    Seconds:

[✓] Keep the values in this object persistent across system shutdowns.

Figure 3-7   Configuring a context scoped business object array

3.6.1 Memory usage

CSBOs can either be held in a database, with a small in-memory cache to hold the most recently used values, or stored completely in memory.

Whether a CSBO is stored in memory or in a database is controlled using the `Keep the values in this object persistent across system shutdowns` check box on the business object page in Event Designer. This is shown on the bottom line of Figure 3-7.

Persistent CSBO

If CSBOs are stored in a database, then the memory requirements are small because the Java heap only contains a small in-memory cache of the most recently used CSBOs. This is the preferred practice.

Non-persistent CSBO

If the CSBOs are held in memory, then 1.5 KB of heap is required for each CSBO, with a further 1.2 KB for each additional array element. The following list gives examples of estimated size requirements:

- If a summary instance CSBO is used to store the average cash withdrawal for 700,000 bank customers, the required Java heap size for the CSBOs would be 1 GB (700,000 x 1.5 KB).
- If a CSBO array was used to store the last five cash withdrawals for 700,000 bank customers, the required Java heap size for the CSBOs would be 4.4 GB (700,000 x 1.5 KB + 700,000 x 4 x 1.2 KB).

Note that non-persistent summary instance CSBO data is never freed from memory unless a server restarts. For CSBO arrays, there will always be a core 1.5 KB for each context, which is never freed from memory unless a server restarts. However, the number of array elements for a particular context can be limited by time or number. For best performance, consider using generational garbage collection with a large enough tenured heap to contain all the CSBO data.

V8.0.1 update: From V8.0.1 onwards, CSBO data can be freed using context management. For further information see the Context Definitions topic in the V8.0.1 IBM Operational Decision Manager Information center.

3.6.2 CSBO performance

As an example of the relative performance of non-persistent and persistent CSBOs, and also to show the relative performance of array and summary instance CSBOs, we ran some test scenarios.

For non-durable event processing our test results showed:

- A complex event rule without a CSBO produces a 42% higher throughput than a complex event rule using a persistent summary CSBO.
- For summary instance CSBOs, non-persistent CSBOs (that is, held only in memory) produce a 30% higher throughput than CSBOs persisted to a database.
- For array CSBOs, non-persistent CSBOs (held only in memory) produce a 60% higher throughput than CSBOs persisted to a database.
- For non-persistent CSBOs, summary instance and small arrays have similar performance.
- For persistent CSBOs, a summary instance produces 25% higher throughput compared to small arrays.

For durable event processing our test results showed:

- A complex event rule without a CSBO produces a 20% higher throughput than a complex event rule using a persistent summary CSBO.
- For summary instance CSBOs, non-persistent CSBOs (held only in memory) produce a 15% higher throughput than CSBOs persisted to a database.

- For array CSBOs, non-persistent CSBOs (held only in memory) produce a 22% higher throughput than CSBOs persisted to a database.
- For non-persistent CSBOs, summary instance and small arrays have similar performance.
- For persistent CSBOs, a summary instance produces 6% higher throughput than small arrays.

Note: The relative throughputs cited show only the relative impact on throughput for the Decision Server Events server. It should be noted that the use of persistent CSBOs also significantly increases the load on the Decision Server Events repository database.

Further information about CSBOs can be found in the Operational Decision Management Information Center:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dserver.events.dev/topics/sharingdataacrossevents.html

Preferred practice: Context scoped business objects

- Unless the number of persistent context scoped business objects is small or the Java heap is very large, the preferred practice is to use persistent CSBO.
- When using non-persistent CSBOs, consider using generational garbage collection with a large enough tenured heap to contain all the CSBO data.

3.7 Multiple event rules and synthetic events

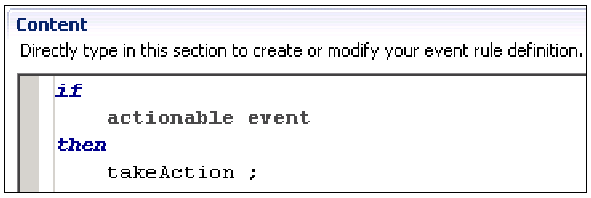
This section describes a number of examples and shows how the choice and complexity of filtering impacts performance. In each case one action is generated for every 100 input events.

3.7.1 Event rule descriptions

The following list describes the event rules and filters used in this comparison:

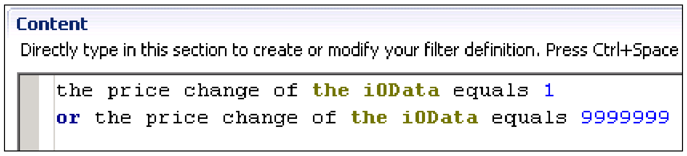
- **One simple event rule**

  This event rule simply checks a value in the input event message to determine whether to send an action. It does not have a context ID. This is the baseline example for comparison with the other examples. Events and actions are both less than 1 KB. The event rule and filter are shown in Figure 3-8 and Figure 3-9.


Content
Directly type in this section to create or modify your event rule definition.

```
if
    actionable event
then
    takeAction ;
```

Figure 3-8 One simple event rule


Content

Directly type in this section to create or modify your filter definition. Press Ctrl+Space

```
the price change of the iOData equals 1
or the price change of the iOData equals 9999999
```

Figure 3-9 The actionable event filter

- **30 simple event rules**

  This example is the same as the one simple event rule case except that 30 filters are used. If each of the 30 filters finds the expected value in the input event, then an action is fired. It clarifies the overhead of evaluating a larger number of simple filters.

- **Synthetic events**

  In this example every input event generates a synthetic event. Synthetic events are internal events that can be defined in Event Designer. They can be fired in the same way as actions, but allow you to define additional event rules to process them. The synthetic event is then processed by the one simple event rule example. This scenario clarifies the performance impact of generating and processing synthetic events.

  For more information about using synthetic events, see the following website:

  http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/index.jsp?topic=/com.ibm.wodm.dserver.events.dev/topics/syntheticevents.html

In our test scenarios, we obtained results that showed the performance of the different filters relative to processing one simple event rule.

For non-durable event processing:

- Evaluating 30 simple event rules compared to one causes less than a 7% reduction in throughput.
- If every event generates a synthetic event, there is an approximately 40% reduction in performance. This occurs because twice as many events are then being processed: the original event and a new synthetic event. If only a small proportion of the initial events generates a synthetic event, the impact will be significantly less.

For durable event processing:

- Evaluating 30 simple event rules compared to one causes less than a 1% reduction in throughput.
- If every event generates a synthetic event, there is an approximately 30% reduction in performance. This occurs because twice as many events are then being processed: the original event and a new synthetic event. If only a small proportion of the initial events generates a synthetic event, the impact will be significantly less.

3.8 Impact of the technology connectors on performance

Decision Server Events can receive events and send actions directly using JMS or indirectly using technology connectors. The use of technology connectors is shown in Figure 3-10.

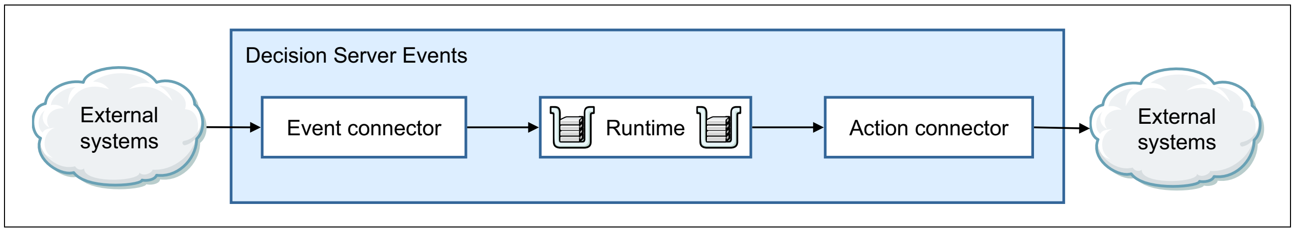


Figure 3-10   The technology connectors

Technology connectors allow Decision Server Events to receive events and send actions to a variety of sources. These sources include databases, FTP, JMS, HTTP, files, and more. They also allow events and actions to be transformed to and from the native Decision Server Event XML format.

The technology connectors provide useful functionality, but also add to the processing overhead. In our test scenario using complex event processing, we obtained results that showed:

For non-durable, non-persistent processing:

- Using a JMS or file event connector has an approximately 30% impact on performance compared to using no connector. Adding an XSLT transformation to the connector introduces an additional 10% overhead.
- Using an HTTP file event connector has a 25% impact on performance compared to using no connector.

For durable, persistent processing:

- Adding a JMS event connector has a 20% impact on performance compared to using no connector. Adding an XSLT transformation to the connector introduces an additional 10% overhead.
- Using the file connector has no noticeable impact on throughput compared to using no connector.

When using the technology connectors, see 4.10, “Technology connectors tuning” on page 50 for information about runtime tuning.

Preferred practice: Technology connectors

Optimal performance is obtained when sending and receiving events and actions directly to and from the Decision Server Events event and action destinations. Use the technology connectors only when required.

4

Decision Server Events: Run time performance tuning

This chapter discusses the tuning of the Decision Server Events run time. It covers considerations that affect your choices at a high level and continues through the tuning aspects of threads, databases, messaging providers, and technology connectors. Monitoring, testing, and diagnosing performance issues round out the topics that are covered on tuning.

The following topics are covered in this chapter:

- High-level considerations for run time tuning
- Tuning the number of event rule processing threads
- The impact of history on performance
- The impact of logging and tracing
- Relative performance of the JMS providers
- Tuning JMS messaging provider: WebSphere Application Server default messaging
- Tuning the JMS messaging provider: WebSphere MQ JMS messaging
- Database tuning
- High availability and high performance
- Technology connectors tuning
- Monitoring performance
- Advice on benchmarking applications
- Diagnosing performance problems

4.1 High-level considerations for run time tuning

This section provides a high-level overview of the factors that affect the run time performance of Decision Server Events. Each of the concepts in the following list is covered with greater detail later in this chapter (or, as noted, elsewhere in the document):

- Ensure that Decision Server Events and the Decision Server Events repository database have sufficient hardware and are not limited by memory, disk, or network resources. See 2.1, “Hardware and monitoring” on page 6.
- Ensure that the JVM is tuned. Consider using generational garbage collection and ensure that the size of the heap is sufficient for the workload. See 2.2, “Tuning the Java virtual machine (JVM)” on page 6.
- Ensure that there are sufficient event rule processing threads. See 4.2, “Tuning the number of event rule processing threads” on page 26.
- Ensure that history is disabled if it is not required. Note that installing the Business Events Tester will automatically enable history. For more information, see 4.3, “The impact of history on performance” on page 28.
- Ensure that tracing and logging are disabled. For more information, see 4.4, “The impact of logging and tracing” on page 31
- Ensure that the appropriate tuning has been applied for your JMS provider. See 4.6, “Tuning JMS messaging provider: WebSphere Application Server default messaging” on page 35 and 4.7, “Tuning the JMS messaging provider: WebSphere MQ JMS messaging” on page 37.
- Ensure that your Decision Server Events repository database is correctly tuned and that the context table is being pruned. See 4.8, “Database tuning” on page 42.
- If you are using WebSphere Application Server default messaging as the JMS provider, and high availability is required, consider using the Scalability with High Availability Topology. See 4.9, “High availability and high performance” on page 43.

4.2 Tuning the number of event rule processing threads

Most event rule processing in the event run time occurs on rule processor threads. The number of these threads is controlled by the following event property:

```
as.director.server.ruleProcessorInstances
```

The following sequence shows where this is configured in the WebSphere Application Server administrative console:

Resources → Resource environment → Resource environment entries → WbeSrv01 → Custom properties

The default value is `31`.

If messages build up on the event destination as the rate of event production increases, then increasing the number of rule processing instances might improve the rate of event processing (by adding greater concurrency). Ensure that the machine has spare CPU capacity to handle the increase in rule processing. Methods of monitoring the event destination are covered in 4.11, “Monitoring performance” on page 54.

When it is running in a cluster, Decision Server Events uses WebSphere eXtreme Scale to route events with a context ID to a particular server in the cluster. To be precise, it routes the events to a specific WebSphere eXtreme Scale partition on a specific server in the cluster. This means that an event with a particular context ID will always be handled by the same rule processing thread and allows data to be cached locally for the context. It also ensures that events with the same context are processed serially, thus preventing any inconsistent, concurrent modification of common data. The distribution of context IDs across the cluster is handled automatically by WebSphere eXtreme Scale.

The optimal number of rule processing threads will be influenced by the number of WebSphere eXtreme Scale partitions used in a cluster (24 by default). You should ensure that the number of rule processing instances is not exactly divisible by either the total number of WebSphere eXtreme Scale partitions, or the number of WebSphere eXtreme Scale partitions on each cluster member. The number of partitions can be confirmed by checking the value of `numberOfPartitions` in the following file:

```
<WODM_HOME>\WAS\AppServer\profiles\WODMSrv01\config\cells\wodmCell\applications\wb
eruntimeear.ear\deployments\wberuntimeear\wberuntime.jar\META-INF\
objectGridDeployment.xml
```

These partitions are spread across the cluster members. With the default total of 24 partitions, if you have two cluster members, there are 12 partitions on each cluster member. With five cluster members, there are five partitions on four servers, and four partitions on the other server.

Failure to ensure that the number of rule processing threads is not exactly divisible by the number of WebSphere eXtreme Scale partitions will not cause any errors. However, it might mean that all `ruleProcessor` threads are not active on all servers, so you might not get the concurrency expected. Therefore, prime numbers make a good choice for the number of `ruleProcessorInstances`.

When you increase the value of this setting, you should also ensure that there are sufficient resources for these threads to use. The following list notes other settings that should be increased in line with this change:

- Events datasource connection pool

  Each rule processor thread could use a database connection from the Event runtime datasource connection pool. The `maximum connections` setting for this connection pool should be set using the following formula as the minimum:

  ```
  history MDB concurrency + value of as.director.server.ruleProcessorInstances + 10
  ```

  This is set in the WebSphere Application Server administrative console using the following menu selections:

  Resources → JDBC → Data sources → Event Runtime Datasource → Connection pools

  To determine the value for `history MDB concurrency`, see 4.3.3, “Tuning history processing” on page 30.

- Events topic onnection factory connection pool

  Each rule processor thread could use a JMS connection from the `WbeTopicConnectionFactory` connection pool. The `maximum connections` setting for this connection pool should be greater than or equal to the following value:

  ```
  value of as.director.server.ruleProcessorInstances + 10
  ```

This is set in the WebSphere Application Server administrative console using the following menu selections:

Resources → **JMS** → **Topic connection factories** → **WbeTopicConnectionFactory** → **Connection pools**

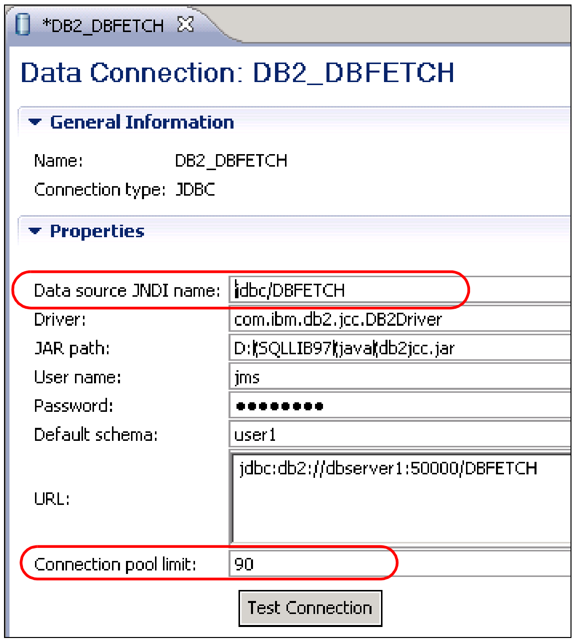
- Data connection connection pool limit

  If you are using a data connection to enrich business objects, you should also increase the connection pool limit. This limit must be set to a value that is greater than or equal to the value of `as.director.server.ruleProcessorInstances`. The methodology for setting the limit depends upon the configuration:

  – If you specify a data source JNDI name in Event Designer on the data connection page, as shown in the first highlight in Figure 4-1, then the connection pool limit is controlled using the connection pool for this data source. This is set in the WebSphere Application Server administrative console using the following menu selections:

    Resources → **JDBC** → **Data sources** → **Event Runtime Datasource** → **Connection pools**

  – If you do not specify a data source JNDI name in Event Designer, then the connection pool limit is controlled by the value that is set in Event designer on the data connection page. See the second highlight in Figure 4-1.



Data Connection: DB2_DBFETCH

General Information

Name:	DB2_DBFETCH
Connection type:	JDBC

Properties

Data source JNDI name:	jdbc/DBFETCH
Driver:	com.ibm.db2.jcc.DB2Driver
JAR path:	D:\SQLLIB97\java\db2jcc.jar
User name:	jms
Password:	••••••••
Default schema:	user1
URL:	jdbc:db2://dbserver1:50000/DBFETCH
Connection pool limit:	90

Figure 4-1 Setting the data connection pool limit

4.3 The impact of history on performance

History is required if you need to run reports or monitor events, actions, filters, or data in real time. When history is enabled, a significant amount of information about events, actions, and filters is written to the Decision Server Events repository.

The history data can be viewed using the reports section of the Decision Server Events administration interface and the Business Space dashboards. You can produce reports that show what events enter Decision Server Events, the actions that are triggered by those events, and the event rules and filters that are evaluated in response to those events. In order to generate these reports, you need to enable history. History is disabled by default unless Business Events Tester is installed. For performance reasons, the installation of Business Event Tester it is not a preferred practice on a production system. History is tracked in history database tables that record events, actions, filters, and other assets that are processed by the Decision Server Events run time.

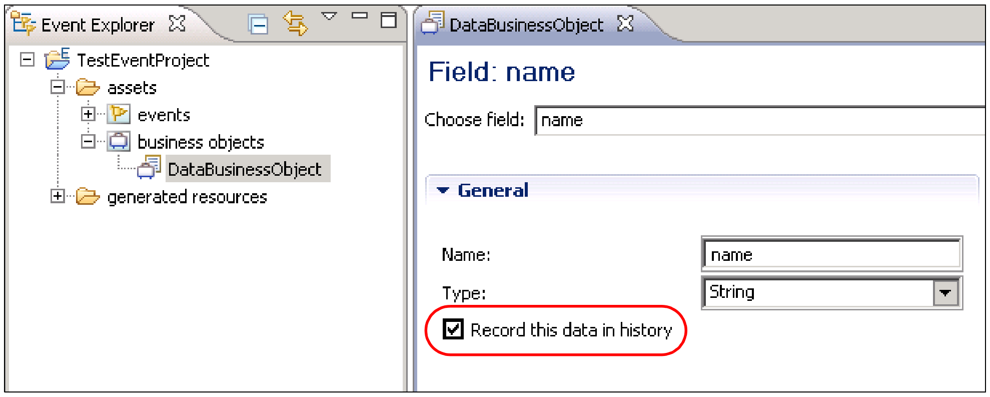
4.3.1 Recording history

For instructions on enabling and disabling history, see the following web page in the Operational Decision Management V8.0 Information Center:

`http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dserver.events.config/topics/tsk_configuring_history_runtime.html`

The business object fields that are recorded in history are set using Event Designer, as shown in Figure 4-2.

Note: These settings will take effect only if history is enabled.



Choose field: name

Name: name

Type: String

[✓ Record this data in history]

Figure 4-2 Setting whether to record a business object field in history

4.3.2 Performance impact of history

As an example, we ran a test scenario using non-durable event processing with no history enabled, and then with history enabled with one business object recorded, and also with five business objects recorded. In our tests, we obtained results showing that:

- The throughput with history enabled and recording one business object field can be 31% of the throughput when history is not enabled.
- The throughput with history enabled and recording five business object fields can be 26% of the throughput when history is not enabled.

Even with just one field selected to record in history, Decision Server Events still records data about all events, filters, rules, and actions. This explains the relatively large performance impact. As can be seen from the data, the number of business object fields that are recorded in history has a relatively small impact on performance.

We ran the same scenario using durable event processing. In our tests, we obtained results showing that:

- The throughput with history enabled and recording one business object field can be 52% of the throughput when history is not enabled.

V8.0.1 update: From V8.0.1 onwards, the impact of using history is significantly improved. In our tests, we obtained results showing that non-durable event processing with history enabled can be a factor of two faster than V8.0.

In V8.0.1, history can also be individually selected on each asset (events, actions, and so on). Selecting a subset of assets can significantly reduce the amount of data recorded and therefore provide further performance gains.

4.3.3 Tuning history processing

If history is enabled and messages are building up on the history topic or queue, you should consider increasing the maximum concurrency on the history activation specification. This will increase the concurrency of Message Driven beans (MDBs) that are writing the history data to the database.

This setting can be altered in the WebSphere Application Server administrative console using the following menu selections:

Resources → JMS Activation specifications → wbe_history

Consider the following information when modifying values using the previous menu selections:

- If you are using WebSphere Application Server default messaging as the messaging provider, select and modify the value of `Maximum concurrent MDB invocations per endpoint`.
- If you are using WebSphere MQ as the messaging provider, select and modify the value of `Maximum server sessions` in the Advanced properties page.

A value of 30 was suitable for the tests run in 4.3.2, “Performance impact of history”. As a part of this change, if you are using WebSphere MQ as the JMS provider, you should also consider modifying the thread pool setting for the WebSphere MQ messaging provider. The modification is needed to account for the extra MDB threads. See 4.7.2, “Tuning the WebSphere Application Server for WebSphere MQ JMS” on page 40 for further details.

Each MDB thread can use a database connection from the Event runtime datasource connection pool. The `Maximum connections` setting for this connection pool is configured in the WebSphere Application Server administrative console using the following menu selections:

Resources → JDBC → Data sources → Event Runtime Datasource → Connection pools

The maximum connections setting should be set to the following minimum standard:

```
history MDB concurrency + value of as.director.server.ruleProcessorInstances + 10
```

The `as.director.server.ruleProcessorInstances` setting is set in the WebSphere Application Server administrative console using the following menu selections:

Resources → Resource environment → Resource environment entries → WbeSrv01 → Custom properties

Preferred practices: history usage

- Carefully consider whether you need to enable history if high performance is required.
- If history is required, consider tuning the MDB concurrency setting.

4.4 The impact of logging and tracing

Logging and tracing have a significant impact on performance and should be minimized wherever possible.

4.4.1 Configuring trace and log files

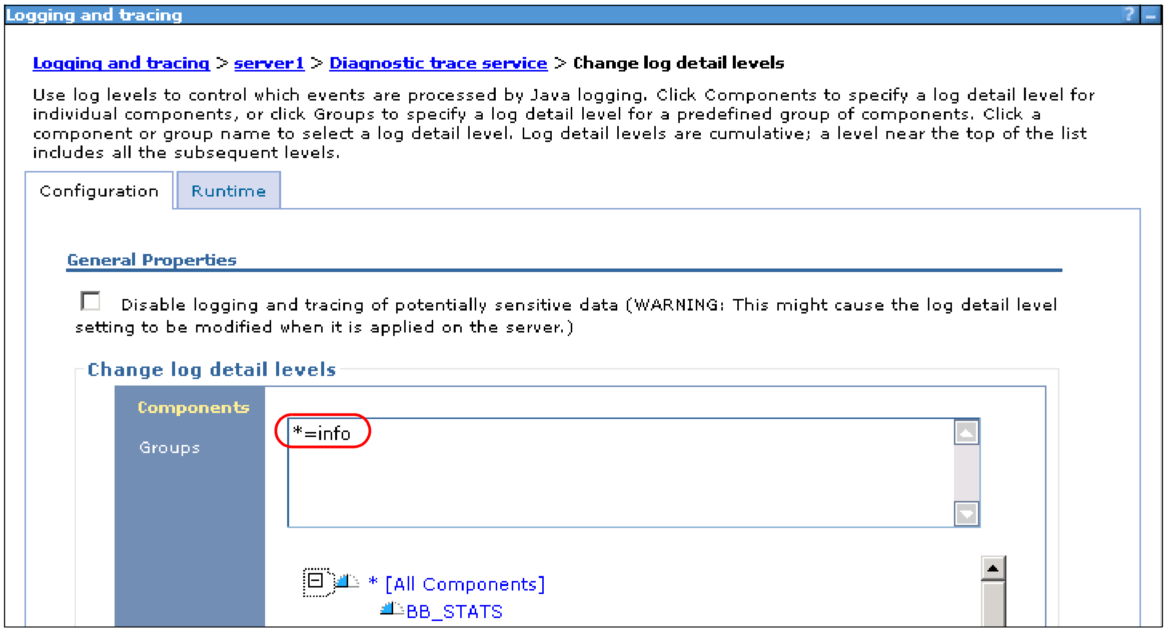
The output files of the trace and log parameters give critical information. It is important to examine these files to ensure that the system is operating correctly and that performance is not being degraded by unnecessary or unexpected log writes. The following list notes how to set the trace and log parameters and where to find the associated output files:

- Application Server diagnostic tracing

  Set the Application Server diagnostic tracing by making the following menu selections:

  Troubleshooting → Logging and Tracing → ***Server1*** **→ Diagnostic Trace service → Change log detail levels**

  Then, set the trace level, as shown in Figure 4-3. The default value of **`*=info`** is an appropriate choice for production use.


Logging and tracing

Logging and tracing > server1 > Diagnostic trace service > Change log detail levels

Use log levels to control which events are processed by Java logging. Click Components to specify a log detail level for individual components, or click Groups to specify a log detail level for a predefined group of components. Click a component or group name to select a log detail level. Log detail levels are cumulative; a level near the top of the list includes all the subsequent levels.

☐ Disable logging and tracing of potentially sensitive data (WARNING: This might cause the log detail level setting to be modified when it is applied on the server.)

*=info

Figure 4-3 Setting the trace level

Setting the trace level enables logging for individual areas of Decision Server Events or WebSphere Application Server. The type and amount of tracing depends on the tracing that is enabled. The following directory is the default log file location:

```
<was_install_dir>\profiles\WODMSrv01\logs\trace.log
```

- Application Server system, error, and ffdc files

  Decision Server Events writes informational and error data to system files. If these files are growing significantly during run time, this will affect performance and can indicate an underlying problem. Check the contents of these files to ensure that there are no configuration problems. The following list shows the default log file locations:

  ```
  <was_install_dir>\profiles\WODMSrv01\logs\SystemOut.log
  <was_install_dir>\profiles\WODMSrv01\logs\SystemErr.log
  <was_install_dir>\profiles\WODMSrv01\ffdc\ (all files)
  ```

- Connector logging when the event connector process is started

  Note: This applies to only the older connectors, which require a separate connector process to be started. It does not apply to the newer HTTP, SOAP, JMS, JDBC, and file connectors, which use the Application Server diagnostic tracing.

  Connector logging is configured in the following location:

  ```
  <DecisionServer_install_dir>\config\wbe\connectorsTrace.properties
  ```

  Setting a value of `com.ibm.wbe.*=debug` results in every event and action packet that is processed by the connectors being written to the connectors log file. The writing of this information has a significant impact on performance. The default setting is `com.ibm.wbe.*=info`, which is generally acceptable for production use. However, some connectors output trace entries. These entries note that they have processed an event or action. This can be useful while debugging or setting up a system, but can have a significant effect on performance. You can check whether the system is receiving a large amount of output from the connectors by looking at the log file. If it contains a large amount of informational trace data that you do not need, change the trace specification to `com.ibm.wbe.*=error`. This setting traces error messages, but does not trace informational messages.

  The default log file location is:

  ```
  <DecisionServer_install_dir>\connectors\logs\connectors.log
  ```

4.4.2 Performance impact of tracing

As an example, we ran a test scenario running without trace enabled (trace string: `*=info`) compared to the same scenario with events user trace enabled (trace string: `*=info: com.ibm.wbe.usertrace.*=finest`). In our tests, we obtained results showing that for non-durable event processing:

- Using the default WebSphere Application Server trace, the throughput was just 10% of the throughput without trace.
- Using high performance extensible logging (HPEL) trace, the throughput was 20% of the throughput with no trace.

HPEL trace is a new feature of WebSphere Application Server V8 that offers improved trace performance. For more information about HPEL, see the WebSphere Application Server information center at the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/ctrb_HPELOverview.html

For durable event processing, our results showed:

- Using the default WebSphere Application Server trace, the throughput was just 30% of the throughput without trace.
- Using high performance extensible logging (HPEL) trace, the throughput was 40% of the throughput with no trace.

If you need to enable tracing to aid in debugging your events project, you should consider increasing both the maximum file size and the maximum number of historical files to ensure that sufficient trace entries are maintained in the `trace.log` file. The default setting might keep only a few seconds worth of tracing (depending on the project and workload).

These settings can be altered in the WebSphere Application Server administrative console using the following menu selections:

Troubleshooting → Logs and Trace → server1 → Diagnostic trace service

Preferred practice: Logging and tracing
Consider the following suggestions when making log and trace decisions:

- Minimize log and trace messages during normal execution.
- If trace is required, then consider increasing the size and number of files.
- For better performance, consider using HPEL trace.

4.5 Relative performance of the JMS providers

Decision Server Events can be configured to use WebSphere Application Server default messaging or WebSphere MQ as the JMS provider. This section discusses the relative performance of these two options.

4.5.1 WebSphere Application Server default messaging vs WebSphere MQ

Although the event run time can use technology connectors to communicate with many sources, all events and actions are ultimately placed on a JMS topic or queue. The only exception to this rule is the case where events are delivered using an external WebSphere eXtreme Scale client. The internal use of WebSphere eXtreme Scale for event distribution is discussed in 4.2, “Tuning the number of event rule processing threads” on page 26. A discussion of external WebSphere eXtreme Scale clients for event delivery, filtering, and caching is outside the scope of this document.

The choice of JMS provider depends on a number of factors. For example, if a company already has a significant investment in a WebSphere MQ infrastructure and skills, then they might prefer to use WebSphere MQ. However, WebSphere Application Server default messaging is an integrated part of Operational Decision Management and does not require any additional product licenses. Another consideration is performance, which is discussed in this section.

As an example, we ran a test scenario to compare the relative performance of the two JMS providers. In our tests, we obtained results showing that for non-durable event processing:

- Using an MQ queue manager co-located with Decision Server Events (using bindings mode), the throughput was 70% of the throughput when using WebSphere Application Server default messaging.
- Using a remote MQ queue manager (using client mode), the throughput was 80% of the throughput when using WebSphere Application Server default messaging.

For durable event processing, our results showed:

- Using an MQ queue manager co-located with Decision Server Events (using bindings mode), the throughput was 90% of the throughput when using WebSphere Application Server default messaging.
- Using an MQ remote queue manager (using client mode), the throughput was 80% of the throughput when using WebSphere Application Server default messaging.

4.5.2 Retrieving events from WebSphere MQ

If an existing WebSphere MQ infrastructure is in place to send events and actions via WebSphere MQ, several options are available to connect to Decision Server Events. Some common configurations are as follows:

- Use WebSphere MQ as the Decision Server Events JMS messaging provider.
  - Co-locate Decision Server Events and WebSphere MQ on the same server and connect using bindings mode.
  - Configure Decision Server Events and WebSphere MQ on separate servers and connect using client mode.
- Use WebSphere default messaging as the Decision Server Events JMS messaging provider.
  - Configure Decision Server Events and WebSphere MQ on separate servers and use a connector to retrieve events from WebSphere MQ using client mode.

As an example, we ran a test scenario to compare the relative performance of these options.

For non-durable event processing, our results showed:

- Using a remote WebSphere MQ queue manager (using client mode) yielded 20% higher throughput compared to using a local WebSphere MQ queue manager (using bindings mode). This is because the remote queue manager is running on its own hardware and, unlike the local case, is not using processing cycles on the Decision Server system.
- Using WebSphere default messaging as the Decision Server Events JMS messaging provider, with a connector to a remote WebSphere MQ queue manager, the throughput was 60% of the throughput using a local WebSphere MQ queue manager (using bindings mode).

For durable event processing, our results showed:

- Using a remote WebSphere MQ queue manager (using client mode), the throughput was 90% of the throughput achieved using a local WebSphere MQ queue manager (using bindings mode).
- Using WebSphere default messaging as the Decision Server Events JMS messaging provider, with a connector, the throughput was 70% of the throughput using a local WebSphere MQ queue manager (using bindings mode).

In both cases, the local queue (using bindings) and the remote queue (using client mode without connectors) offer significant performance advantages over the connector. Note that the remote queue option has introduced additional hardware by running WebSphere MQ remotely. This explains the 20% performance gain when using remote versus local queue managers for non-durable processing.

4.6 Tuning JMS messaging provider: WebSphere Application Server default messaging

WebSphere Application Server default messaging offers two choices of message store. Message store is where persistent and hardened, non-persistent messages are stored. Non-persistent messages can be hardened (written to disk) to free memory when messages start to build up on a destination. The message store types are:

- A file-system-based file store
- A database-backed implementation called data store

By default, Decision Server Events uses the file store to store persistent messages. WebSphere Application Server also provides the ability to use data store implementation to store persistent messages.

The benefits of each approach are discussed in the WebSphere Application Server information center. Refer to the entry on *Relative advantages of a file store and a data store* at the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/cjm0001_.html

In our tests, the performance of Decision Server Events, using either file store or data store with a remote IBM DB2® database, was similar for both persistent and non-persistent messaging.

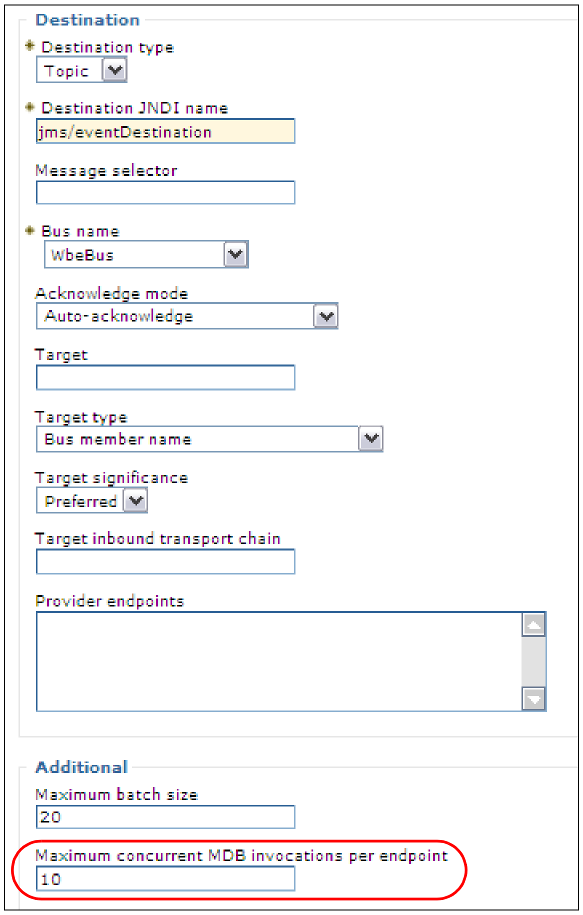
4.6.1 Concurrent processing: WebSphere Application Server default messaging

For non-durable processing, events on the event destination are processed by an MDB using the `wbe_events` activation specification. By default, this activation specification has the `maximum concurrent MDB invocations per endpoint` set to `10`, which means that a maximum of 10 events can be processed concurrently. If events build up on the event destination, consider increasing the `maximum concurrent MDB invocations per end point` for the `wbe_events` activation specification.

For durable event processing, if events are building up on the durable event destination, consider increasing the `maximum concurrent MDB invocations per end point` for the `wbe_events_durable` activation specification.

These parameters can be set using the WebSphere Application Server administrative console, as shown in Figure 4-4 on page 36. Use the following menu selections to access that configuration point:

Servers → Resources → JMS → Activation specifications → wbe_events (or wbe_ wbe_events_durable)


Destination

Destination type: Topic

Destination JNDI name: jms/eventDestination

Message selector:

Bus name: WbeBus

Acknowledge mode: Auto-acknowledge

Target:

Target type: Bus member name

Target significance: Preferred

Target inbound transport chain:

Provider endpoints:

Additional

Maximum batch size: 20

Maximum concurrent MDB invocations per endpoint: 10

Figure 4-4 Setting MDB concurrency

4.6.2 Miscellaneous WebSphere Application Server default messaging tuning

When using non-persistent messaging, adjusting the WebSphere Application Server default messaging setting can improve throughput. The following setting is particularly useful when there is a buildup of messages on the JMS topics or queues:

```
sib.msgstore.omTemporaryStoreCacheSize=60000
```

For persistent messaging, the following setting can improve throughput, particularly when there is a buildup of messages on the JMS topics or queues:

```
sib.msgstore.omPermanentStoreCacheSize=60000
```

To configure these settings, make the following menu selections:

Service integration → Buses → WbeBus → [Topology] Messaging engines → engine_name → [Additional Properties] Custom properties

For more information about these settings, refer to *Controlling the memory buffers used by a messaging engine* in the WebSphere Application Server V8.0 information center:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/tjm0220_.html

If you see disk contention when using WebSphere Application Server default persistent messaging, you might want to move the message store to a faster disk location. To move locations, follow the instructions in *Modifying file store configuration* in the WebSphere Application Server V8.0 information center at the following address:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjm1030_.html

Note that you will need to recreate the `WbeTopicSpace` destination after the file store location has been moved.

4.7 Tuning the JMS messaging provider: WebSphere MQ JMS messaging

This section is only intended as a general guide for WebSphere MQ tuning. Refer to the WebSphere MQ information center and WebSphere MQ performance reports for more details and specific tuning suggestions.

The WebSphere MQ information center for WebSphere MQ is found at the following website:

http://www-01.ibm.com/software/integration/wmq/library/index.html

The WebSphere MQ performance reports and tuning is found at the following website:

https://www-304.ibm.com/support/docview.wss?uid=swg27007150

4.7.1 Tuning the queue manager

This section discusses the tuning of the WebSphere MQ queue manager.

Log files

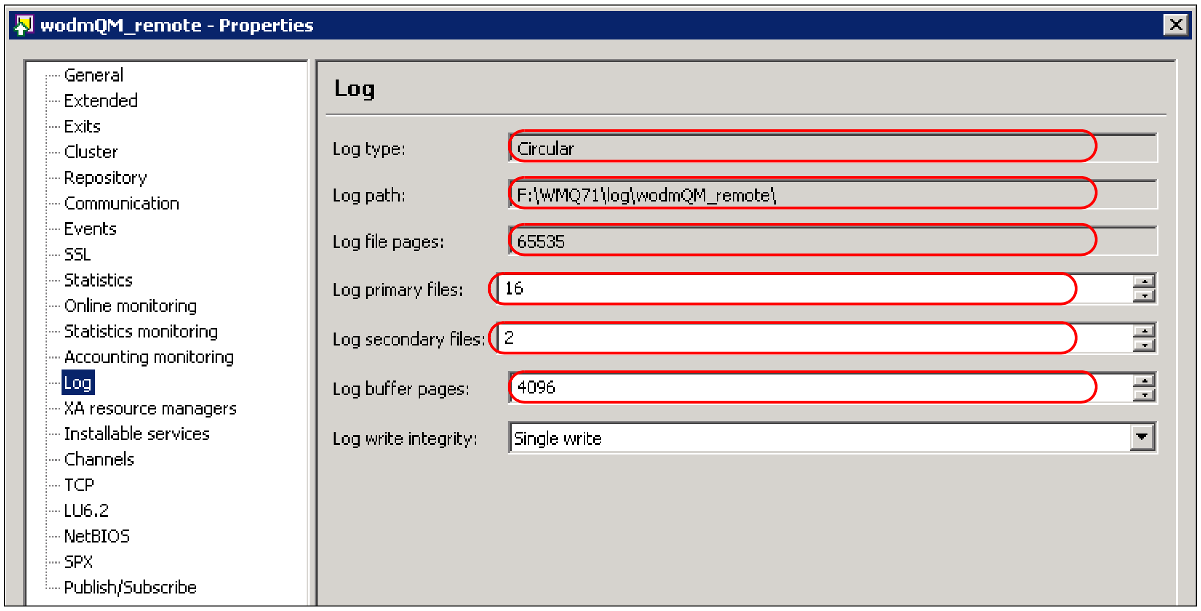
One of the key considerations, especially for persistent processing, is the configuration of the queue manager log files. Incorrectly tuned log files will limit the speed at which messages can be persisted to disk and will therefore limit the overall message throughput. The following list identifies the important tuning parameters to consider:

- Log path: Place the data and log files on their own fast disk subsystems.
- Log type: Circular logging provides optimal performance.
- Log file pages: If there is sufficient disk space, allocate the maximum size.
- Number of primary and secondary log files: Log primary at value `16` and the secondary at value `2` as a reasonable starting point.
- Log buffer pages: Use the largest value possible for optimal performance.

For further details about these settings, see the article on *Configuring and tuning WebSphere MQ performance on Windows and Linux* at the following website:

http://www.ibm.com/developerworks/websphere/library/techarticles/0712_dunn/0712_dunn.html

These settings are configured from WebSphere MQ Explorer by right-clicking the **Queue manager** and selecting **Properties**, as shown in Figure 4-5 on page 38.


wodmQM_remote - Properties

Log

Log type:	Circular
Log path:	F:\WMQ71\log\wodmQM_remote\
Log file pages:	65535
Log primary files:	16
Log secondary files:	2
Log buffer pages:	4096
Log write integrity:	Single write

Figure 4-5 Configuring log file settings

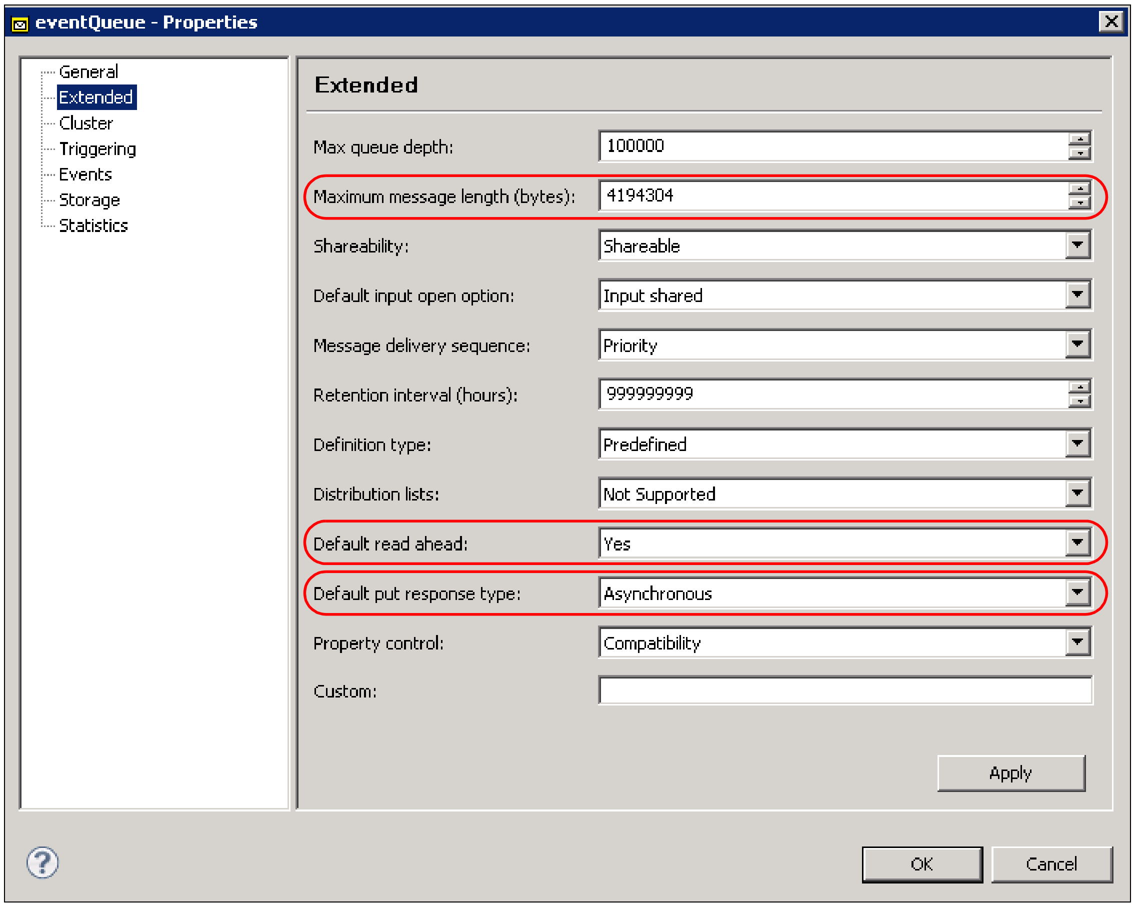
General considerations

The following list presents general performance considerations for WebSphere MQ:

- The maximum queue depth for the Decision Server Events queues. Ensure that this is large enough to support the largest expected runtime queue depth. This setting is configured from WebSphere MQ Explorer by right-clicking **Queue** and selecting **Properties** → **Extended.**

  This step is shown in Figure 4-6 on page 39.

- For the non-persistent, non-durable event and action destinations, consider using `Read ahead` and `Asynchronous put response type`. Read ahead improves performance by sending messages to a client in advance of an **`MQGET`** request. Asynchronous puts improve performance by streaming messages without waiting for individual replies. Both of these are relevant to client connections where a high quality of service is not required. This is also shown in Figure 4-6.


eventQueue - Properties

Extended

Max queue depth:	100000
Maximum message length (bytes):	4194304
Shareability:	Shareable
Default input open option:	Input shared
Message delivery sequence:	Priority
Retention interval (hours):	999999999
Definition type:	Predefined
Distribution lists:	Not Supported
Default read ahead:	Yes
Default put response type:	Asynchronous
Property control:	Compatibility
Custom:	

Figure 4-6 MQ queue properties

- If you are running in a stable environment, without user exits, consider making channels trusted (set `MQI bind type` to `fast`). This improves performance by reducing path length. MQI bind type is set from WebSphere MQ Explorer by right-clicking **Queue manager** and selecting **Properties** → **Channels**.
- Consider the optional channel setting of `Channels\SHARECNV=1`. This configuration ensures that producers and consumers have their own sockets, which can be beneficial for performance. This setting is configured from WebSphere MQ Explorer by selecting the **Channels folder**, right-clicking the relevant channel and selecting **Properties** → **Extended**. Set the `Sharing conversations` value to `1`.
- Ensure that the `Max Channels` and `Max active channels` settings for the queue manager are greater than the sum of all the WebSphere Application Server session pools. This prevents contention over channels and can therefore improve performance. Setting a high value of `5000` should be sufficient. These settings can be configured from WebSphere MQ Explorer by right-clicking the **Queue manager** and selecting **Properties**. Select **Channels** and set the `Max channels` and `Max active channels` to `5000`.
- Consider setting the `DefaultPQBufferSize` and `DefaultQBufferSize` in the `qm.ini` file. The following syntax shows both tuning parameters:

  ```
  DefaultPQBufferSize=16777216
  DefaultQBufferSize=16777216
  ```

  This `DefaultQBufferSize` setting will allow 16 MB of memory for non-persistent messages. After the memory limit is exceeded, the MQ server will spill additional non-persistent messages to disk and performance is degraded.

This `DefaultPQBufferSize` setting will allow 16 MB of persistent messages (per queue) to be cached in memory. When the limit is exceeded, the MQ server will need to access the disk to read the messages and performance is degraded.

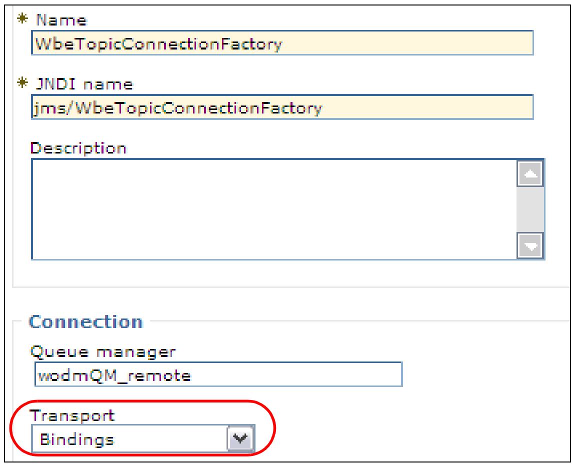
Note: These buffer sizes are a *per queue* setting, so each queue in the queue manager will use 16 MB of memory. This value should be adjusted to fit the workload, the number of queues, and the memory available.

4.7.2 Tuning the WebSphere Application Server for WebSphere MQ JMS

This section discusses the tuning of the WebSphere Application Server to optimize the interaction with WebSphere MQ.

Bindings mode

If Decision Server Events and WebSphere MQ are located on the same machine, ensure that the application server is using the bindings transport to connect to WebSphere MQ. This allows WebSphere MQ to use the Java Native Interface (JNI) to call directly into the queue manager API, rather than using communication through the network. This provides significantly better performance. You can set the transport on the connection factory in the WebSphere Application Server administrative console, as shown in Figure 4-7. To set the transport, select **Resources** → **JMS** → **TopicConnectionFactory** → **WbeTopicConnectionFactory** and specify `Bindings` for `Transport`.


* Name: WbeTopicConnectionFactory

* JNDI name: jms/WbeTopicConnectionFactory

Description:

Connection

Queue manager: wodmQM_remote

Transport: Bindings

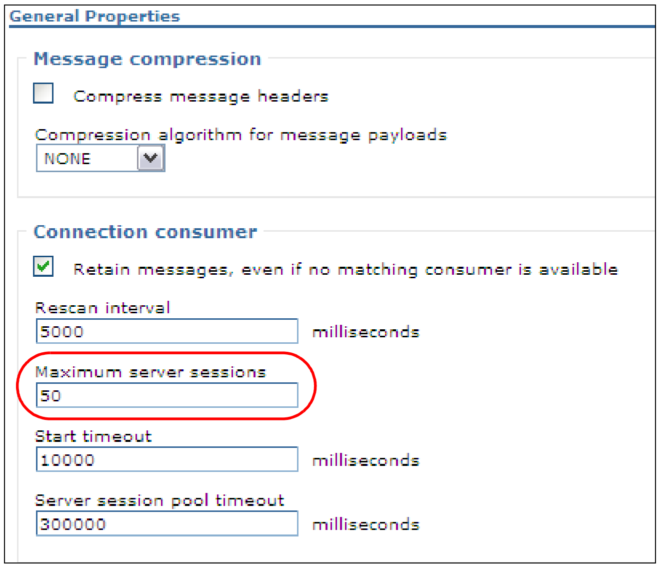
Figure 4-7 MQ transport setting configured for Bindings

Concurrent processing: WebSphere MQ

For non-durable processing, events on the event destination are processed by an MDB using the `wbe_events` activation specification. By default, this activation specification has the `maximum server sessions` set to `10`. This value means that a maximum of 10 events can be processed concurrently. If events build up on the event destination, consider increasing the maximum server sessions for the `wbe_events` activation specification.

Similarly, for durable event processing, if events are building up on the durable event destination, consider increasing the maximum server sessions for the `wbe_events_durable` activation specification.

These parameters can be set using the WebSphere Application Server administrative console, as shown Figure 4-8. The parameters can be accessed by selecting **Servers** → **Resources** → **JMS** → **Activation specifications** → **wbe_events (or wbe_ wbe_events_durable)** → **Advanced properties**.


General Properties

Message compression

☐ Compress message headers

Compression algorithm for message payloads
NONE

Connection consumer

☑ Retain messages, even if no matching consumer is available

Rescan interval
5000 milliseconds

Maximum server sessions
50

Start timeout
10000 milliseconds

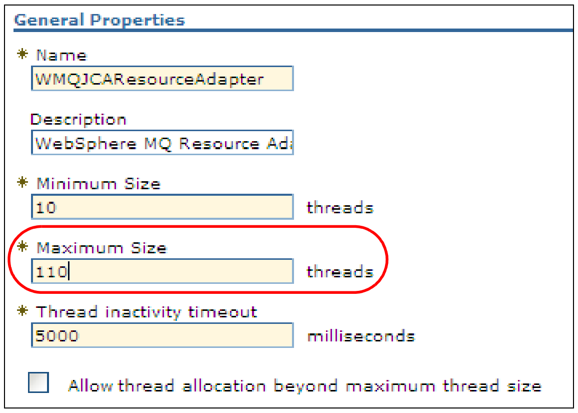
Server session pool timeout
300000 milliseconds

Figure 4-8 Setting maximum server sessions

Note that each of the server sessions must obtain a thread from the `WMQJCAResourceAdapter` thread pool in order to run. If the thread pool is not sized correctly, the server session will block waiting for a thread, which will reduce concurrency and performance. Too few threads might also cause work to be rejected and time out errors in the WebSphere Application Server `SystemOut.log`.

To ensure that this pool is large enough, add up the `maximum server sessions` properties for each WebSphere MQ activation specification on the application server. If Decision Server Events has 11 activation specifications with 10 server sessions each, this would require a minimum thread pool size of 110. Any increase in maximum server sessions should be matched by an increase in the number of threads in the thread pool.

To change the setting from the administrative console, as shown in Figure 4-9, select **Servers** → **Server types** → **WebSphere Application Servers** → **server1** → **Threadpools** → **WMQJCAResourceAdapter**.


General Properties

* Name
WMQJCAResourceAdapter

Description
WebSphere MQ Resource Ad

* Minimum Size
10 threads

* Maximum Size
110 threads

* Thread inactivity timeout
5000 milliseconds

☐ Allow thread allocation beyond maximum thread size

Figure 4-9 Setting MQ JCA thread pool size

Queue and topic connection factories

Ensure that the maximum number of connections (default 10) correctly reflects the maximum number of connections expected for each of the connection factories. These can be set by selecting **Resources** → **JMS** → **Topic connection factories** → **WbeTopicConnectionFactory** → **Connection pools** → **Maximum connections**.

To obtain additional information about tuning the number of connections for the WbeTopicConnectionFactory, see 4.2, "Tuning the number of event rule processing threads" on page 26.

Additional tuning information

Additional tuning information about WebSphere Application Server activation specifications and WebSphere MQ is located at the following website:

`http://www.ibm.com/developerworks/websphere/library/techarticles/1110_titheridge/1110_titheridge.html?ca=drs-`

Preferred practices: WebSphere MQ tuning

The following list summarizes the important considerations in queue manager tuning:

- Tune the logging.
- Ensure that the queue depths are set appropriately.
- For non-durable processing, consider using asynchronous puts and read ahead.
- Tune the size of the Default[P]QBufferSize memory buffers.
- Use trusted channels where applicable.

The following list summarizes the important considerations in WebSphere Application Server tuning for MQ:

- Use bindings mode if possible.
- Ensure that there are enough server sessions and `WMQJCAResourceAdapter` threads.

4.8 Database tuning

This section discusses the general principles of database tuning for Decision Server Events.

4.8.1 General tuning

The exact tuning of the database depends upon the type of database, the hardware, and the nature of the workload. However, the following list gives some high-level preferred practices:

- Consider using fast disk subsystems (RAID/SAN) for the database. Ideally, place data and logs on different disk subsystems.
- Monitor the disk, memory, network, and cpu utilization of the database to identify any potential bottlenecks.

4.8.2 Database entry management

Event rules using a context ID need to access the Decision Server Events repository database to record events in the context or steps table and, for complex events, also need to retrieve information about past events. Although recently used steps data is cached in memory, the cache has a limited size and, in many cases, the required information needs to be retrieved from the database. If the amount of data in the steps table becomes too large (and large in this case might be many millions of entries), then the performance can be adversely affected.

It is therefore a preferred practice to periodically prune the steps table to remove unwanted entries. For more Information on pruning the steps table, refer to pertinent topic in the Operational Decision Management information center at:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/index.jsp?topic=/com.ibm.wodm.dserver.events.admin/topics/pruningthewbestepstable.html

V8.0.1 Update: From V8.0.1 onwards, context data can be freed using context management. For further information, see the *Context Definitions* topic in the V8.0.1 Operational Decision Manager information center.

Preferred practices: Decision Server Events database tuning
The following list summarizes the important factors to consider in Decision Server Events database tuning:

- Use fast disks.
- Monitor the CPU, disk, network, and memory usage to identify bottlenecks.
- Ensure that the database is correctly tuned for the workload.
- Periodically prune the Decision Server Events repository steps table, either automatically or manually.

4.9 High availability and high performance

Decision Server Events uses WebSphere Application Server clustering to provide high availability and scalability. There are many different ways to configure WebSphere Application Server clustering for messaging. This section covers some of the most common configurations employed when using WebSphere Application Server default messaging or WebSphere MQ as messaging provider.

4.9.1 High availability and performance when using WebSphere Application Server default messaging

Two of the most important factors affecting cluster performance are messaging engine policy and cluster topology. These factors are discussed in this section.

Messaging engine policy

The choice of messaging engine policy has a significant impact on cluster performance. The following list describes the available choices:

- High availability: The high availability messaging engine policy configures a single messaging engine for the cluster. The messaging engine will fail over to any of the other application servers in the cluster.
- Scalability: The scalability policy configures a messaging engine for each server in the cluster. This can provide better performance because the messaging load is spread across all the servers in the cluster.

- Scalability with high availability: The scalability with high availability messaging engine policy configures a messaging engine for each server in the cluster. Each messaging engine can also fail over to one other server in the cluster. This configuration provides for high availability and scalability.

The scalability and scalability with high availability policies provide better performance than the high availability policy. This is because with these policies there are multiple messaging engines to support the messaging load. High availability topology uses a single messaging engine.

More information about message engine policies can be found in the section on message engine policy assistance in the WebSphere Application Server information center at the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/cjt1005_.html

Cluster configuration topologies

There are two main cluster configuration topologies possible when using WebSphere Application Server default messaging as the messaging provider. These cluster configuration topologies are the silver topology and the gold topology.

Silver topology

The silver topology consists of a single cluster of WebSphere Application Server instances. Each instance contains both a messaging engine (ME) and the Decision Server Events run time. For more information about this topology refer to the following website:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/index.jsp?topic=/com.ibm.wodm.dserver.events.config/topics/creatingandconfiguringasilvertopologycluster.html

The silver topology configuration is shown in Figure 4-10.

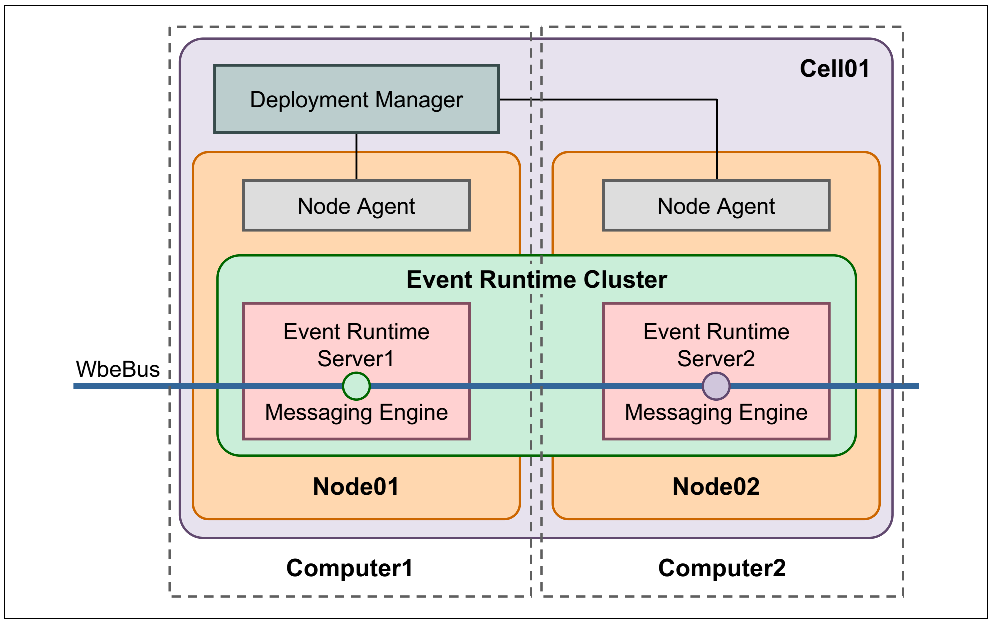


Figure 4-10 Overview of silver topology

Gold topology

The gold topology consists of two separate clusters of WebSphere Application Server instances (one for the run time and one for the messaging). The messaging and events runtime clusters can be on either the same or different hardware. For more information about this topology, refer to the following website:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/index.jsp?topic=/com.ibm.wodm.dserver.events.config/topics/creatingandconfiguringagoldentopologycluster.html

The configuration for gold topology is shown in Figure 4-11.

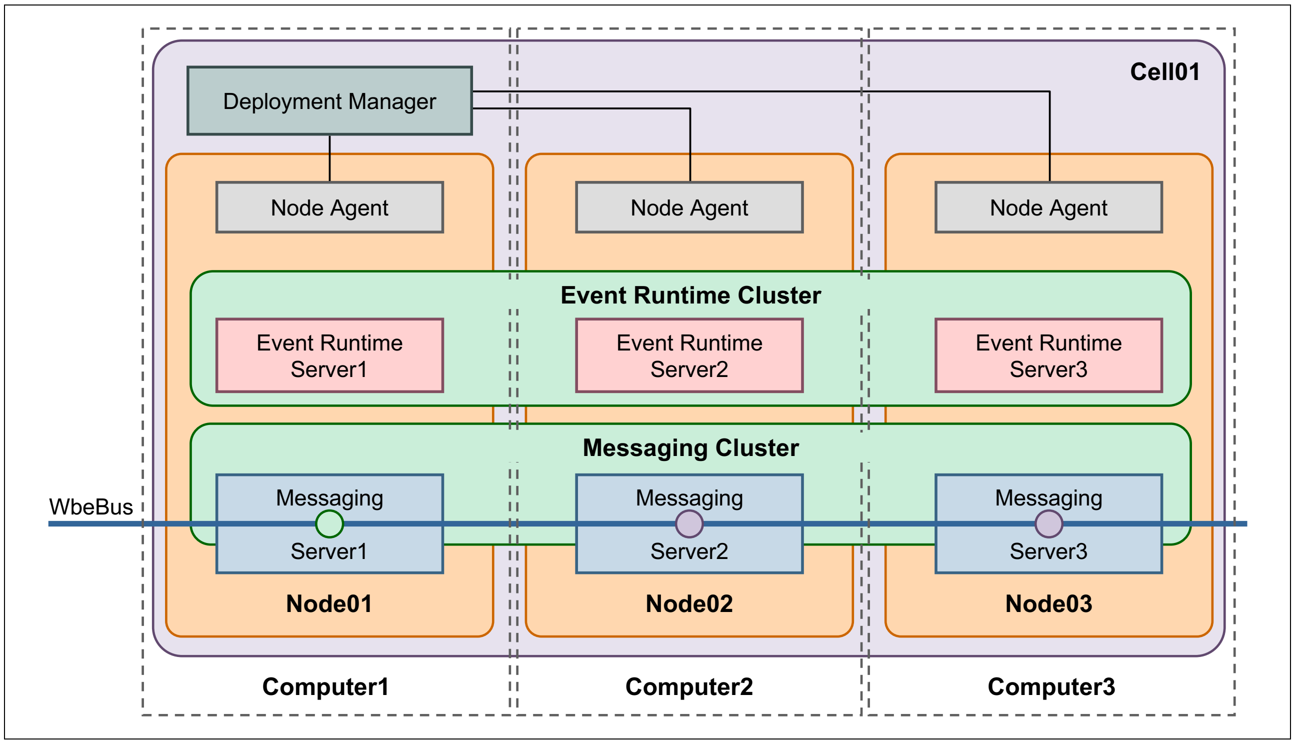


Figure 4-11 Overview of gold topology

Performance comparison of silver and gold topologies

We ran a test scenario to compare the relative performance of the silver and gold topologies. Each topology was configured on a three-node cell, using the scalability with high availability messaging engine policy. The gold topology had a messaging server and an events runtime server on each node, whereas the silver topology had an events runtime server that contained a messaging engine on each node. In our tests, we obtained results showing:

- For non-durable event processing, the gold topology achieved 20% lower throughput than the silver topology.
- For durable event processing, the gold topology achieved 30% lower throughput than the silver topology.

The relative throughput will vary depending on persistence, workload, and configuration. In most cases, the silver topology will provide better throughput. It is more efficient for Decision Server Events to receive a message from a local messaging engine, within the same application server, than from a messaging engine in a different application server process.

4.9.2 High availability and performance when using WebSphere MQ as the messaging provider

The default configuration, when using WebSphere MQ as a JMS provider in a cluster, is to have a single WebSphere MQ queue manager and a cluster of Decision Server event runtime servers. See Figure 4-12.

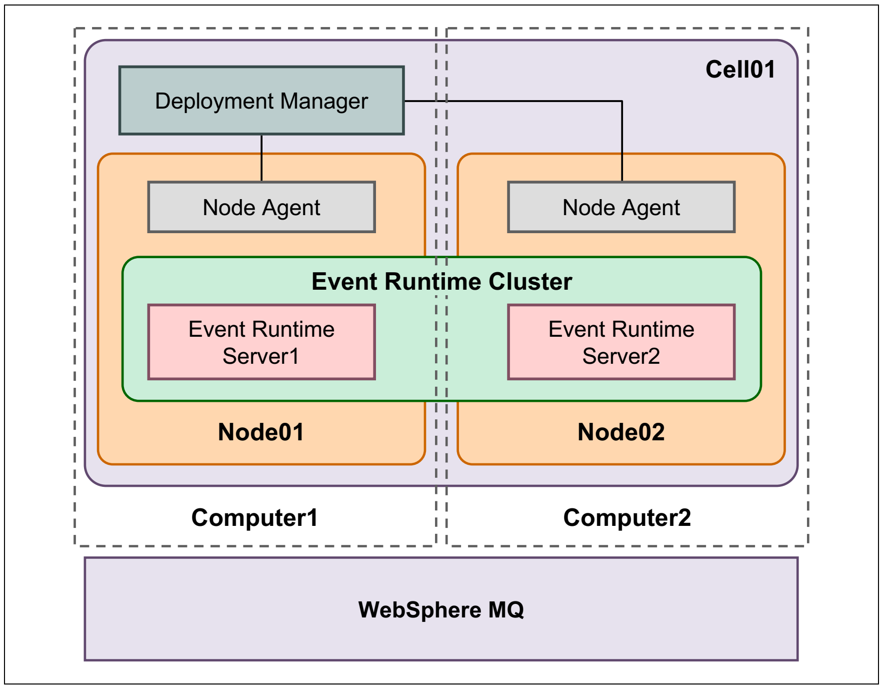


Figure 4-12 A Decision Server Events cluster using WebSphere MQ as JMS provider

This configuration requires a single queue manager in a WebSphere MQ instance. The single queue manager can be made highly available using HA clusters or multi-instance queue managers.

This configuration is suitable for most installations and scaling can be achieved by adding more event runtime cluster members. However, in a high performance environment, it is possible that the single queue manager might not be able to support the required throughput. To resolve the throughput issue, some manual configuration is required to configure the Decision Server Events runtime cluster to consume from multiple queue managers. An example of this configuration, shown in Figure 4-13, requires one queue manager per WebSphere MQ instance.

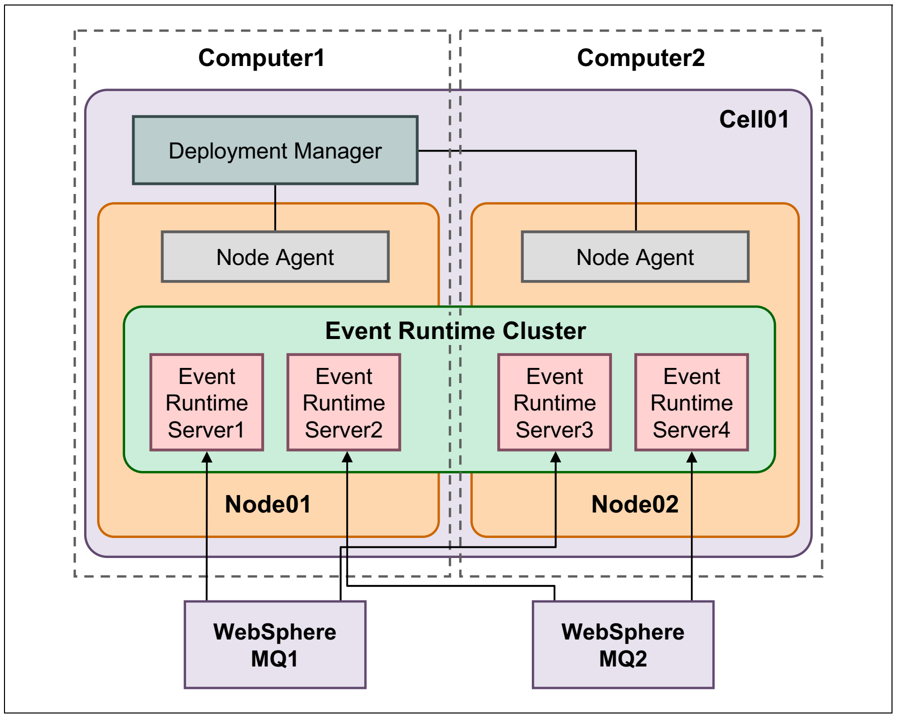


Figure 4-13 A Decision Server Events cluster using two WebSphere MQ queue managers

The key point about this configuration is that there are two event runtime servers on each Decision Server Events runtime node. Each event runtime server is configured using server-scoped activation specifications and queue connection factories. The servers are configured to receive events from separate WebSphere MQ queue managers. This means events will continue to be processed even if a piece of hardware in the configuration fails.

Configuration of this topology is accomplished using the following steps:

1. Define new activation specifications at each server scope that point to the required queue manager for that server. In Figure 4-13, server1 and server3 point to WMQ1, and server2 and server4 point to WMQ2. These new activation specifications should be copies of the existing `wbe_events` and `wbe_events_durable` activation specifications that are at cell scope. They should have different names, such as `wbe_events_server1` and `wbe_events_durable_server1`, but the JNDI names must be set to `jca/wbe_events` and `jca/wbe_events_durable`.
2. Define new queue connection factories at each server scope that point to the required queue manager for that server. These new queue connection factories should be copies of the existing `WbeQueueConnectionFactory` that is at cell scope. They should have different names, such as `WbeQueueConnectionFactory _server1`, but the JNDI names must be set to `jms/WbeQueueConnectionFactory`.
3. Synchronize the nodes and restart the servers. Decision Server Events run time is now configured to receive events from two queue managers.

The topics, `actionTopic` and `durableActionTopic`, are still hosted on one queue manager for simplicity. However, they can also be split across multiple queue managers. This is done using a similar technique to define the topic connection factory (`WbeTopicConnectionFactory`) and the following activation specifications at the server scope:

`wbe_reset_watch, wbeca_http_as, wbeca_jms_as, wbeca_soap_as, wbeca_file_as, wbeca_jdbc_as.`

When creating these activation specifications, be sure to copy all the settings from the original activation specification.

If you are using the history capability of Decision Server Events, you should also define the `wbe_history` activation specification at the server scope.

4.9.3 Tuning for large clusters

The Decision Server Events run time uses WebSphere eXtreme Scale to route events with a context ID to the correct server in the cluster. To perform this routing, WebSphere eXtreme Scale uses partitions that are spread across the cluster. The default total number of partitions is 24. This should be suitable for clusters with up to six event runtime servers. For clusters with more than six event runtime servers, consider increasing the number of partitions to a minimum of four partitions per server (10 servers should have a minimum of 40 partitions). You should also allow for any future expansion of your events runtime cluster because modifying the number of partitions requires a full cell outage.

The number of WebSphere eXtreme Scale partitions is defined in a file called `objectGridDeployment.xml`. That file is located in the `wberuntime.jar` file, which is contained within the `wberuntimeear.ear`.

To modify this setting, perform the following steps:

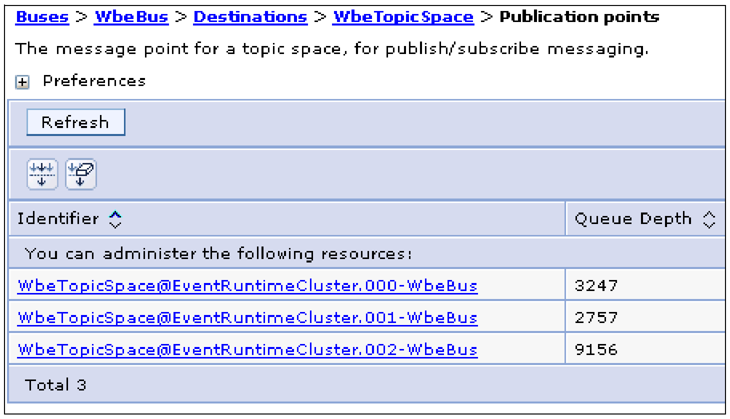
1. Locate the current version of the `wberuntimeear.ear` (by default it is located in `<WODM_HOME>/director/lib`). Make sure that you take a copy of the original file.
2. Extract the `wberuntime.jar` file from the copied `wberuntimeear.ear` file:
   ```
   jar -xf wberuntimeear.ear wberuntime.jar
   ```
3. Extract the `objectGridDeployment.xml` file from the `wberuntime.jar` file:
   ```
   jar -xf wberuntime.jar META-INF/objectGridDeployment.xml
   ```
4. Edit the `objectGridDeployment.xml` file and change the attribute, `numberOfPartitions` in the `mapSet` element, to your new number of partitions. The resulting entry should be similar to the following syntax:
   ```
   <mapSet name="mapSet" numberOfPartitions="40" minSyncReplicas="0"
   maxSyncReplicas="0" maxAsyncReplicas="0" numInitialContainers="1">
   ```
5. Save the changes.
6. Update the `wberuntime.jar` with the new `objectGridDeployment.xml` file:
   ```
   jar -uf wberuntime.jar META-INF/objectGridDeployment.xml
   ```
7. Update the `wberuntimeear.ear` file with the modified `wberuntime.jar` file:
   ```
   jar -uf wberuntimeear.ear wberuntime.jar
   ```
8. In the WebSphere Application Server admin console, update the `wberuntimeear.ear` application using the following steps:
   a. Select **Applications** → **Application Types** → **WebSphere enterprise applications**.
   b. Select the **wberuntimeear** application, and click **Update**.
   c. Enter the path to the updated `wberuntimeear.ear` file. Click **Next**.
   d. Default options (Fast Path) is fine here. Click **Next**.
   e. Click **Step 3 summary** and click **Finish**.
9. After the update is done, synchronize all nodes and restart both the deployment manager and all application servers.

4.9.4 Disabling the WatchReset Message Driven Bean

If using WebSphere Application Server default messaging in a cluster, the `WatchReset MDB` can cause unnecessary messaging traffic in the cluster. This MDB is only required if you are using track events. For more information about track events, see the following website:

`http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/index.jsp?topic=/com.ibm.wodm.dserver.events.dev/topics/whatarewatchevents.html`

If you are not using track events and notice a buildup of messages, consider disabling the `WatchReset MDB`. To see whether messages are building up, check the `actionTopic` publication points in the administration console. This view is accessed by selecting **Service integration** → **Buses** → **WbeBus** → **Destinations** → **WbeTopicSpace** → **Publication points** (see Figure 4-14).


Buses > WbeBus > Destinations > WbeTopicSpace > **Publication points**

The message point for a topic space, for publish/subscribe messaging.

Identifier	Queue Depth
You can administer the following resources:	
WbeTopicSpace@EventRuntimeCluster.000-WbeBus	3247
WbeTopicSpace@EventRuntimeCluster.001-WbeBus	2757
WbeTopicSpace@EventRuntimeCluster.002-WbeBus	9156
Total 3	

Figure 4-14 Messages building up on actionTopic publication point

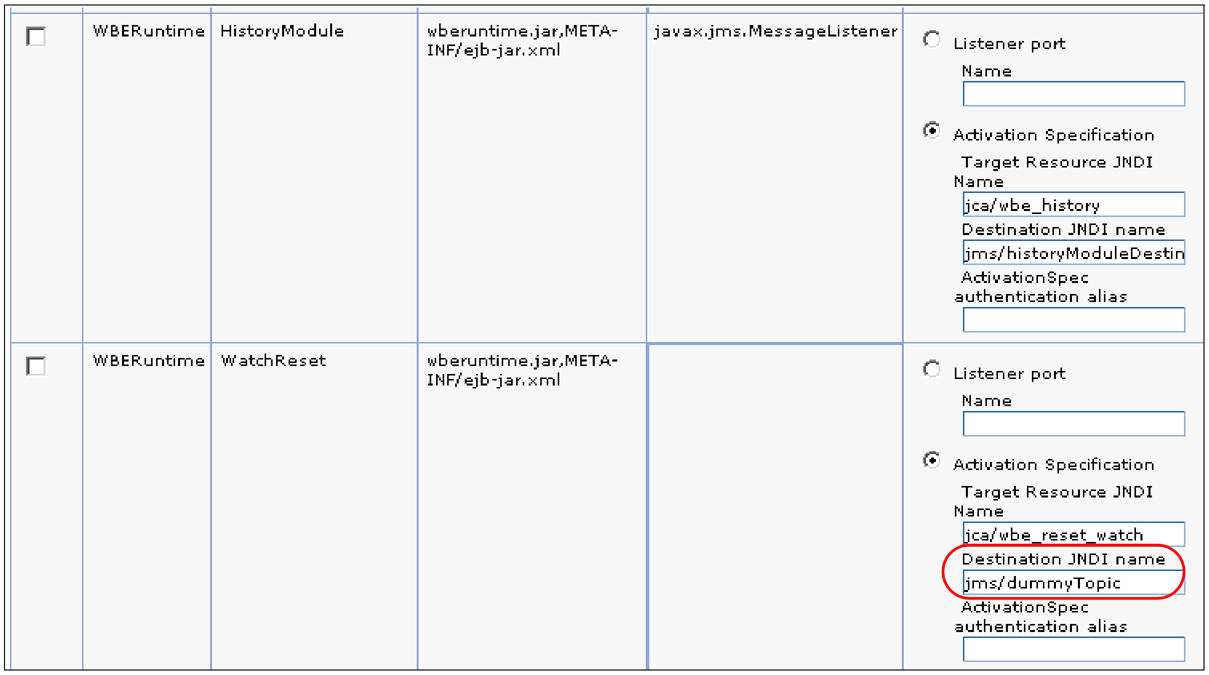
To disable the `WatchReset MDB`, perform the following steps:

1. Define a new JMS topic called `dummyTopic`.
   a. Using the administration console, navigate to **Resources** → **JMS** → **Topics**, and click **New** with the scope set to `Cell`.
   b. Make the appropriate name settings:
      - Name: `dummyTopic`
      - JNDI name: `jms/dummyTopic`
      - Topic name: `dummyTopic`
      - Bus name: `WbeBus`
      - Topic space: `WbeTopicSpace`
   c. Click **OK**.
2. Set the `wbe_reset_watch` activation specification to use the new `dummyTopic`.
   a. Using the administration console, navigate to **Resources** → **JMS** → **Activation specifications**, and click `wbe_reset_watch`.
   b. Change `Destination JNDI name` to `jms/dummyTopic`.
   c. Click **OK**.
3. Set the `WatchReset MDB` to use the new `dummyTopic`.
   a. Using the administration console, navigate to **Applications** → **Application types** → **WebSphere enterprise applications** and click `wberuntimeear`.

b. Under Enterprise Java Bean Properties, click `Message driven bean listener bindings`.

c. Change the `Destination JNDI name` setting for the `WatchReset` bean to `jms/dummyTopic` (See Figure 4-15).

d. Click **OK**.



☐	WBERuntime	HistoryModule	wberuntime.jar,META-INF/ejb-jar.xml	javax.jms.MessageListener	○ Listener port Name: ● Activation Specification Target Resource JNDI Name: jca/wbe_history Destination JNDI name: jms/historyModuleDestin ActivationSpec authentication alias:
☐	WBERuntime	WatchReset	wberuntime.jar,META-INF/ejb-jar.xml		○ Listener port Name: ● Activation Specification Target Resource JNDI Name: jca/wbe_reset_watch Destination JNDI name: jms/dummyTopic ActivationSpec authentication alias:

Figure 4-15 Setting the Destination JNDI name for the WatchReset bean

4. Save the changes to the configuration, synchronize the nodes, and restart the cell to acknowledge the changes.

Preferred practices: High availability and high performance

- Use the silver topology for best performance if you are using WebSphere Application Server default messaging as the JMS provider.
- Use the scalability with high availability topology if high availability and scalability are required.
- Consider increasing the number of `Object Grid` partitions if you have more than eight event runtime cluster members.

4.10 Technology connectors tuning

The technology connectors for JMS, HTTP, SOAP, JDBC, and File System all run as applications within the WebSphere Application Server environment. Running as an application allows the connectors to benefit from the high availability and scalability provided by WebSphere Application Server. This section covers some tuning considerations for these technology connectors.

4.10.1 Increasing the concurrency of the action connectors

When using the JMS, HTTP, SOAP, JDBC, or File System action connectors, the connector runs as a Message Driven Bean (MDB) within the WebSphere Application Server. The MDBs retrieve messages from the action topic or durable action topic and then process the action before sending it to the relevant location. The concurrency of these connectors is controlled using the activation specifications. If you are using one of these connectors and observe messages building up on the action topic, it is possible that increasing the concurrency will improve the rate at which actions are processed. Methods of monitoring the action topic are covered in 4.11, “Monitoring performance” on page 54. Table 4-1 identifies the utilized activation specifications.

Table 4-1 Action connector activation specifications

Action connector	Activation specification
JMS	`wbeca_jms_as`
HTTP	`wbeca_http_as`
SOAP	`wbeca_soap_as`
JDBC	`wbeca_jdbc_as`
File System	`wbeca_file_as`

To modify the concurrency setting, using the WebSphere Application Server administrative console, navigate to **Resources** → **JMS** → **Activation specifications**. Click the activation specification you want to modify. The following list gives configuration points based on your messaging provider:

- If you are using WebSphere Application Server default messaging, modify `Maximum concurrent MDB invocations per endpoint`.
- If you are using WebSphere MQ as messaging provider, modify the value of `Maximum server sessions` on the Advanced properties page.

Note: If you are using the JMS event connector, the concurrency is also controlled by the activation specification you define on the event connector page in `Event Designer`. Therefore, concurrency of both activation specifications must be reviewed.

4.10.2 Using a selector to retrieve actions from the action topic

If you use your own JMS client to receive actions generated by the events run time, then by default, the subscriber will receive all actions that are published to the action topic. This will include all the expected actions and actions that were generated for use by the connectors. This can result in your JMS client receiving a significant number of unwanted JMS messages and adds unnecessary load on the messaging system.

It is possible to receive only actions that the external subscriber is interested in by using a selector when creating the subscriber. All actions are published with a JMS property of `actionName=` *`<The action name>`*.

For example, to configure your system so that a selector only receives actions, you can create a subscription using your own JMS client, called `myAction`. Use the following syntax:

```
consumer = session.createConsumer(actionTopic, actionName=myAction);
```

Use the following syntax for a durable subscriber:

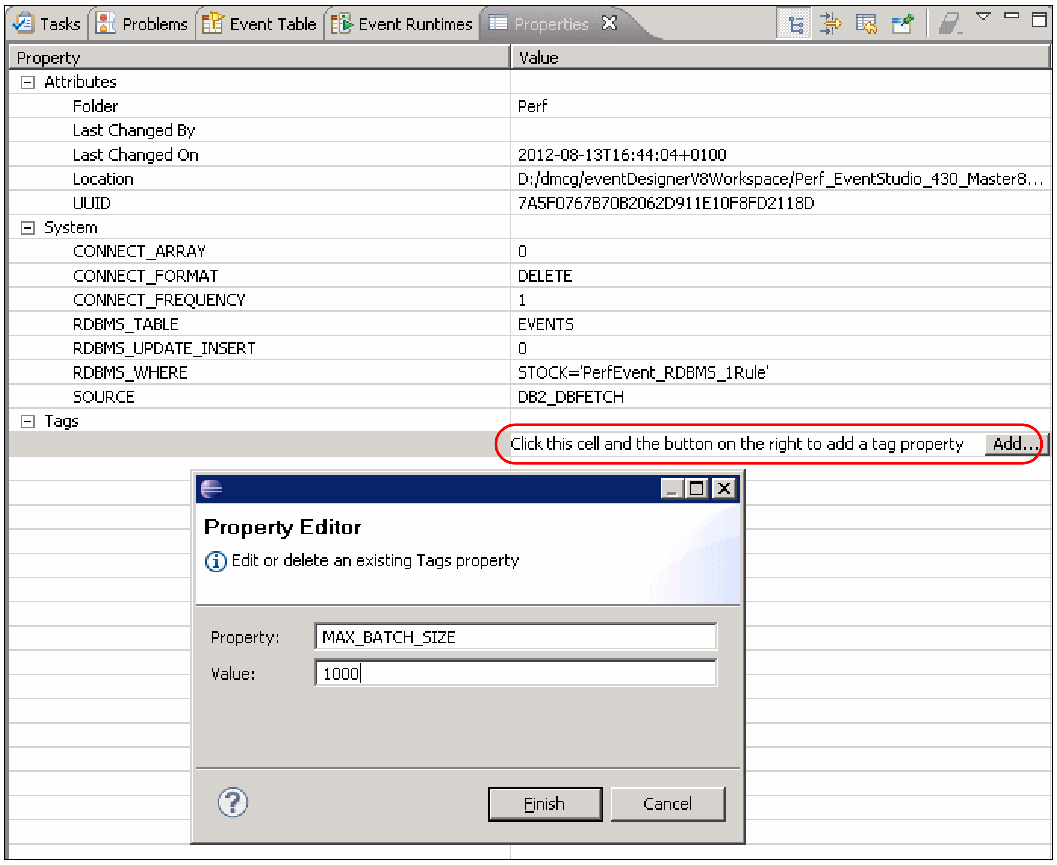
```
consumer = createDurableSubscriber(actionTopic, subID, actionName=myAction, 
false);
```

4.10.3 Controlling the batch size for the JDBC event connector

When using the JDBC connector to consume events from a database, you can control how many rows are processed in a single transaction by setting the batch size. The default batch size is `500`, which means that within a single transaction 500 rows of the database table are processed and sent to the event run time event destination. This should be suitable for most applications. However, if the database operation takes a long time, you can improve performance (particularly when using either a `Select & Delete` or `Select & Update` with a synchronization column operation) by increasing the batch size. This reduces the number of database operations. The JMS events, sent by the connector to the event destinations, are not committed until the batch is complete. By setting the batch size too high, you risk overloading the event destinations by placing a large number of messages on the destination at one time.

Take the following steps to modify the batch size:

1. Click the event in Event Designer.
2. Use the menu bar to select **Window** → **Show view** → **Properties**.
3. Click the cell as shown in Figure 4-16, then click **Add**.



Property	Value
Attributes	
Folder	Perf
Last Changed By	
Last Changed On	2012-08-13T16:44:04+0100
Location	D:/dmcg/eventDesignerV8Workspace/Perf_EventStudio_430_Master8...
UUID	7A5F0767B70B2062D911E10F8FD2118D
System	
CONNECT_ARRAY	0
CONNECT_FORMAT	DELETE
CONNECT_FREQUENCY	1
RDBMS_TABLE	EVENTS
RDBMS_UPDATE_INSERT	0
RDBMS_WHERE	STOCK='PerfEvent_RDBMS_1Rule'
SOURCE	DB2_DBFETCH
Tags	
	Click this cell and the button on the right to add a tag property Add...

Property Editor

Edit or delete an existing Tags property

Property: MAX_BATCH_SIZE

Value: 1000

Figure 4-16 Increasing the batch size for the JDBC event connector

4. Set the property as `MAX_BATCH_SIZE` and the value to the batch size required.

5. Click **Finish**.
6. Save the event, then deploy the project to the run time.

4.10.4 Controlling the File System event connector event delivery rate

By default, the File System connector processes lines from a file as fast as possible. This can result in the connector sending events to the run time at a faster rate than the run time can process them. Events sent at this pace can lead to a buildup of messages on the JMS event destinations, which can affect performance. Other types of events might have to wait in the queue before they are processed.

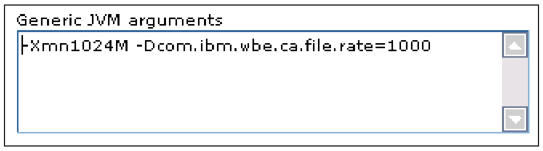
It is possible to control the rate at which the File System connector sends events to the run time using a JVM property. The rate is specified in events per second. The connector will aim to meet this target but it is not a guaranteed send rate.

When using the File System connector, if you notice the event destinations are filling up, consider changing the event delivery rate to limit the rate of production. The rate should be set to a value below the rate at which the run time can process events. Making this parameter change will avoid events building up on the JMS event destination.

Use the following steps to set this parameter:

1. In the WebSphere Application Server administrative console, navigate to **Servers → Server Types → WebSphere application servers → server1 → Java and Process Management → Process definition → Java Virtual Machine**.
2. As shown in Figure 4-17, add the following to the `Generic JVM arguments`:

   ```
   -Dcom.ibm.wbe.ca.file.rate=1000
   ```


Generic JVM arguments

```
-Xmn1024M -Dcom.ibm.wbe.ca.file.rate=1000
```

Figure 4-17 Setting the File System connector event delivery rate

3. Click **Apply**, then save the change to the configuration.
4. In a cluster, synchronize the change to the nodes.
5. Restart all servers to acknowledge the change.

4.10.5 Distributing events across a cluster when using the JDBC and File System technology connectors with WebSphere Application Server default messaging

The JDBC and File System event connectors run on a single cluster member at a time. However, they will fail over to another server if the cluster member they are running on stops. If you are using WebSphere Application Server default messaging with the silver topology (discussed in “Silver topology” on page 44), this can result in all events produced by the JDBC and File System event connectors being consumed by just one cluster member. If you are using the JDBC or File System event connectors in a cluster and observe that one server is significantly busier than the others, then you might benefit from configuring WebSphere

Application Server to balance messages across all available queue points. For more information see *Workload sharing with queue destinations* at:

`http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cjt0014_.html`

Take the following steps to configure this setting using the WebSphere Application Server administrative console:

1. Navigate to **Resources** → **JMS** → **Queues** and click `eventQueue`.
2. Under `Control across multiple queue points per Message Producer` set the Local queue point preference to `Do not prefer a local queue point over other queue points`.
3. Repeat step 2 for the `durableEventQueue`.
4. Click **Apply**, then save the changes to the configuration.
5. Restart all cluster members so that the change takes effect.

4.11 Monitoring performance

This section discusses the tools that are available for performance monitoring.

4.11.1 Performance Monitoring Infrastructure (PMI)

Decision Server Events has a number of built-in PMI statistics to help users understand and monitor the performance of their applications. These statistics can help answer the following questions:

- What event types are arriving, and at what rates?
- What event rules and rule groups are being executed?
- What actions are being generated?
- At what rates are these things occurring and how many times have they occurred since monitoring was started?
- Are you using the level of JMS persistence you are expecting?
- Are you processing events fast enough, or are they building up on queues or topics?

If Decision Server Events is using default messaging as the JMS provider, then additional information about the queues and topics will also be available.

Further information about PMI statistics can be found in the *Operational Decision Management information center*:

`http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/index.jsp?topic=/com.ibm.wodm.dserver.events.monitoring/topics/monitoringbyusingpmi.html`

4.11.2 Service Integration Bus Performance tool

The Service Integration Bus Performance tool automatically starts and dynamically displays the relevant, key performance statistics for Decision Server Events users. It is an IBM developerWorks® offering, rather than an IBM product. This offering has been used by many customers during their testing phase to provide a real-time understanding of Decision Server Events performance. To access the tool see the following website:

https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=13a714e5-6379-4325-aeee-c0c88ec3d15b

4.11.3 Service Integration Bus Explorer tool

The Service Integration Bus Explorer tool can be used to view and manage messaging resources in a Service Integration Bus. To access the tool see the following website:

https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=fe21f954-66dd-40c2-bfc4-45eb0b7d51eb

4.11.4 WebSphere MQ Explorer Tool

When using WebSphere MQ as the JMS provider, the JMS performance data can be monitored with the standard WebSphere MQ tool known as WebSphere MQ Explorer.

4.11.5 Hardware monitoring

Standard operating system tools (such as, **`perfmon`** for Windows and **`iostat/vmstat`** for Linux and UNIX) can be used to monitor disk activity, CPU usage, and memory usage. Monitoring these areas helps you to determine whether the disk subsystem's processing speed or paging is limiting performance.

4.12 Advice on benchmarking applications

When evaluating the performance of Decision Server Events, it is important to use a representative workload. Review the following list for important considerations:

- As with all Java performance measurements, ensure that the JIT has compiled the Java code before making any measurements. You can achieve this by running several thousand representative events through Decision Server Events after starting it for the first time.
- Ensure that you use a realistic number of context IDs for your expected production workload. This ensures that the database activity, in storing event context information and retrieving related events, is representative.
- Ensure that the input workload is not overloading the Decision Server Events input queues or topics. If the input rate is too high, events back up onto the input destinations until they are full, and you will not get accurate measurement of steady state performance.
- Monitor the CPU, network, disk, and memory utilization to ensure the throughput is not being constrained by hardware limitations.
- Ensure the Decision Server Events and database servers are tuned in accordance with this paper and the product documentation.

The advice provided in this document is based on measurements taken for certain workloads on certain hardware. Although the advice should be valid for most Decision Server Events workloads on most operating systems and hardware, you should always verify any changes in a suitable performance measurement environment before deploying them in production.

Preferred practices: Performance measurement

- Ensure that the Java code on the server is JIT-compiled.
- Use a realistic number of context IDs.
- Ensure that measurements are steady state, that is, that the JMS event and actions queues and topics are not overloaded.
- Monitor system performance to identify any system limitations, including CPU, memory, disk, and network limitations.

4.13 Diagnosing performance problems

This section shows how to identify and resolve performance bottlenecks within Decision Server Events.

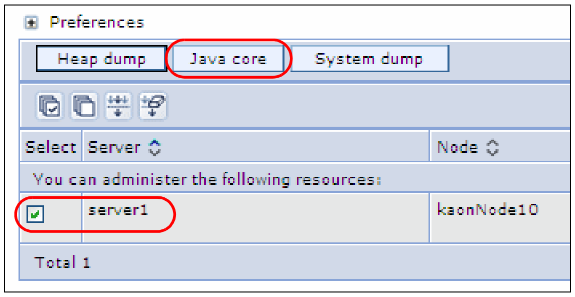
4.13.1 Obtaining diagnostic information

The first step in the process is to obtain the necessary diagnostic information for problem analysis and resolution.

Generating Java cores

A Java core is an important source of information that contains a snapshot of thread activity, locks, class loading, and limited information about memory usage. Java cores can be generated using the WebSphere Application Server administrative console and following these steps:

1. Select **Troubleshooting** → **Java dumps and cores.**
2. To generate a Java core, select the check box next to the server name and click the **Java core** button, as shown in Figure 4-18.



Select	Server	Node
You can administer the following resources:		
✓	server1	kaonNode10
Total 1		

Figure 4-18 Generating Java cores

This will cause a Java core file, `javacore.date.time.processid.javacoreNumber.txt`, to be generated in the `profile root` directory of the selected server. The following example shows sample syntax:

```
<WODM_HOME>\WAS\AppServer\profiles\WODMSrv01\bin\javacore.20120829.132204.6932.000
1.txt
```

For diagnostic purposes, it is preferred practice to take three or more Java cores at least 30 seconds apart. This will provide different snapshots for analysis and comparison.

Problem determination, messaging queue and topic depths

The event destination and action topic queue depths can be monitored. The following list notes two products that can be used:

- Performance Monitoring Infrastructure statistics for WebSphere default messaging
- WebSphere MQ Explorer tool for WebSphere MQ

These are discussed in the section on “Monitoring performance” on page 54.

System performance

System performance (disk activity, CPU, and memory) can be monitored using standard operating system tools such as **perfmon** for Windows and **iostat/vmstat** for Linux and UNIX.

4.13.2 The number of messages on the event destination is growing over time

If the number of messages on the event destination is continually growing, and there is spare CPU capacity on the Decision Server Events server, the most likely cause is a shortage of event rule processing threads. See “Tuning the number of event rule processing threads” on page 26.

There might be other causes, which can be investigated by generating a Java core, as described in the previous section. Open the Java core file and search the thread stacks for the following text strings to see how many threads are engaged in a given activity:

- `SlaveEngine.processNext`

  These are the event rule processing threads. For each thread in the Java core with this method, review the top few items in the stack to see the current activity of the rule processing threads.

  If 20% or more of the rule engine threads have `java/lang/Object.wait` as the top of the stack, with `JobQueue.read` further up the stack, this indicates that there are idle event rule processing threads available to do work. See Figure 4-19 for an example.

```
Java callstack:
at java/lang/Object.wait(Native Method)
at java/lang/Object.wait(Object.java:196(Compiled Code))
at EDU/oswego/cs/dl/util/concurrent/LinkedQueue.poll(Bytecode PC:101(Compiled Code))
at com/ibm/wbe/utils/LinkedQueue.poll(LinkedQueue.java:86(Compiled Code))
at com/ibm/wbe/utils/JobQueue.read(JobQueue.java:320(Compiled Code))
at com/ibm/wbe/server/engine/SlaveEngine.processNext(SlaveEngine.java:404(Compiled Code))
```

Figure 4-19 Java call stack for an idle rule processing thread

  No additional event rule processing threads should be required. This assumes that the total number of event rule processing threads is not exactly divisible by either of the following numbers:

  – The total number of WebSphere eXtreme Scale partitions on each cluster member

  – The number of WebSphere eXtreme Scale partitions on each cluster member

  This division was discussed in “Tuning the number of event rule processing threads” on page 26. This assumption is true in a default configuration. If you have changed the number of event rule processing threads, or the number of WebSphere eXtreme Scale partitions, ensure that you have not violated this principle. If this principle is not adhered to, some event rule processing threads will always be idle and you will not achieve optimal concurrency.

If all of the rule engine threads are busy, but are not engaged in the same activity, consider increasing the number of event rule processing threads. For reference, this concept is discussed in the section on "Tuning the number of event rule processing threads" on page 26.

If a large number of threads (90% or more) are involved in the same activity, this suggests that activity is responsible for the performance bottleneck. For example, if 90% of the threads are making JDBC calls to a database, then the database is likely to be the performance bottleneck. Consider tuning or increasing the capacity of the database. Increasing the number of event rule processing threads might also improve performance.

- `EventReceiverBean.onMessage`

  These are the Message Driven Bean (MDB) threads that remove the events from the event destination and pass them to the event rule processing threads.

  If the number of threads containing this method is equal to MDB concurrency, then increasing the MDB concurrency is likely to improve performance. Information about MDB concurrency can be found in two sections:

  – For WebSphere Application Server default messaging and MDB, see "Concurrent processing: WebSphere Application Server default messaging" on page 35.

  – For WebSphere MQ and MDB, see "Concurrent processing: WebSphere MQ" on page 40.

  If the top of the stack of any of these MDB threads is `java/lang/Thread.sleep`, then the event rule processing threads are overloaded and are temporarily unable to accept events from the MDBs. See Figure 4-20.

```
Java callstack:
at java/lang/Thread.sleep(Native Method)
at java/lang/Thread.sleep(Thread.java:851(Compiled Code))
at
com/ibm/wbe/server/engine/objectgrid/ObjectGridHelper.dispatch(ObjectGridHelper.java:284(Compi
led Code))
at com/ibm/wbe/server/engine/MasterEngine.dispatch(MasterEngine.java:787(Compiled Code))
at com/ibm/wbe/server/EventProcessor.process(EventProcessor.java:385(Compiled Code))
at
com/ibm/wbe/server/DurableEventPersistenceImpl.eventReceiverQueueWrite(DurableEventPersistence
Impl.java:70(Compiled Code))
at com/ibm/wbe/server/ReceiverBeanHelper.ProcessMessage(ReceiverBeanHelper.java:206(Compiled
Code))
at ejbs/EventReceiverBean.onMessage(EventReceiverBean.java:55(Compiled Code))
```

Figure 4-20 Java call stack for MDBs when event rule processing threads are busy

Overloading might occur if you send a large number (thousands) of events with the same context ID in quick succession. In this case, if possible, ensure your context IDs have a better distribution. Overloading can also signify that the event rule processing threads are unable to cope. In that case, analyze the event rule processing threads as discussed previously to find the reason why they are unable to cope.

4.13.3 Other potential performance issues

The following list notes miscellaneous performance issues that you might encounter and tuning options:

- The number of messages on the action topic is growing over time.

  This can happen when using connectors and indicates additional tuning is required. See the section on “Increasing the concurrency of the action connectors” on page 51.

  If using WebSphere Application Server default messaging in a cluster, then consider disabling the **`WatchReset`** Message Driven Bean. For details, see 4.9.4, “Disabling the WatchReset Message Driven Bean” on page 49.

- The number of messages on the history destination is growing over time.

  Review the tuning advice in the section on “Tuning history processing” on page 30.

- Messages such as `Failed to dispatch event after 600 retries due to PersistentEngineQueue full exception` appear in the `SystemOut.log` file.

  If this is seen in the `SystemOut.log` files, then the event rule processing threads are overloaded and are temporarily unable to accept more work. Analyze the event rule processing threads as discussed previously. Analyzing will help identify the reason why threads are unable to cope.

5

Decision Server Rules: Rule Designer tuning

Rule Designer is the development environment for business rule applications. It is integrated into Eclipse. Developers can take advantage of this integration to develop their Java projects with rule projects. Developers and architects can therefore use Rule Designer to integrate and deploy the business rule application into their enterprise client application.

Architects, developers, and business analysts use Rule Designer to design the business rule application, author, review, and debug business rules. Rule Designer also has tools for keeping the rules synchronized with the Decision Center repository.

The following topics are covered in this chapter:

- Tuning of build
- Tuning of the synchronization
- Guidelines for benchmarking an application
- Analyzing the rule project characteristics

5.1 Tuning of build

Here are some suggestions to optimize the build performance for rules:

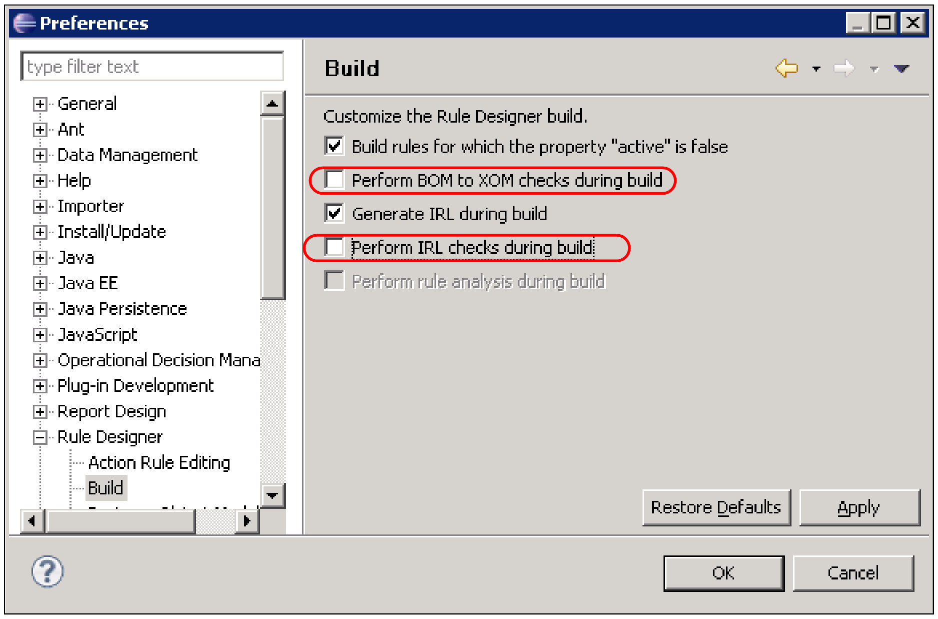
- For large rule projects, turn off the automatic build feature:

  Select **Rule Designer Menu** → **Project** and clear the `Build Automatically` flag.
- Turn off the BOM to XOM and IRL checks:

  Select **Rule Designer Menu** → **Windows** → **Preferences** → **Rule Designer** → **Build**. Then, clear the following options:

  – `Perform BOM to XOM checks during build`
  – `Perform IRL checks during build`

Clearing these options provides optimal performance. However, you lose the rule analysis of your rule set. See Figure 5-1.



Build

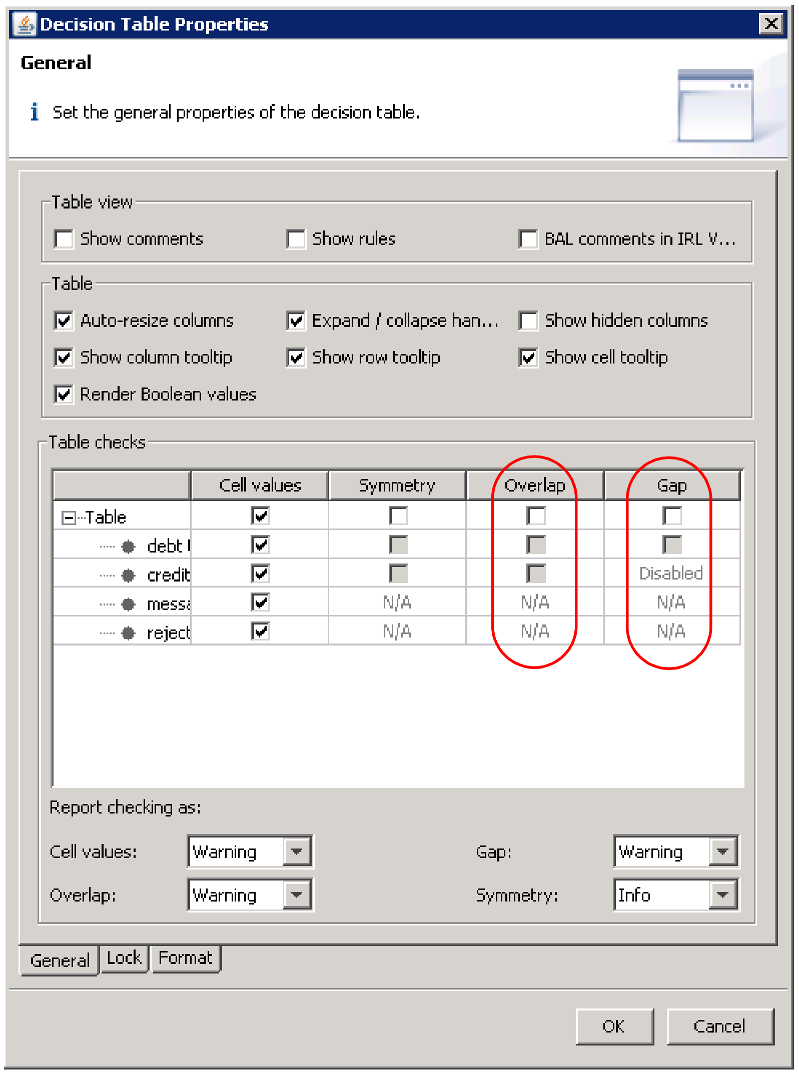
Customize the Rule Designer build.

- [✓] Build rules for which the property "active" is false
- [ ] Perform BOM to XOM checks during build
- [✓] Generate IRL during build
- [ ] Perform IRL checks during build
- [ ] Perform rule analysis during build

Figure 5-1 Build preferences

Decision tables can be optimized use the following steps:

1. Right-click a decision table to access its properties.
2. Remove the `Table Checks` on Overlap and Gap, as shown in Figure 5-2 on page 63.


Decision Table Properties

General

Set the general properties of the decision table.

Table view

- [ ] Show comments
- [ ] Show rules
- [ ] BAL comments in IRL V...

Table

- [✓] Auto-resize columns
- [✓] Expand / collapse han...
- [ ] Show hidden columns
- [✓] Show column tooltip
- [✓] Show row tooltip
- [✓] Show cell tooltip
- [✓] Render Boolean values

Table checks

	Cell values	Symmetry	Overlap	Gap
Table	✓	☐	☐	☐
debt l	✓	☐	☐	☐
credit	✓	☐	☐	Disabled
messa	✓	N/A	N/A	N/A
reject	✓	N/A	N/A	N/A

Report checking as:

Cell values:	Warning	Gap:	Warning
Overlap:	Warning	Symmetry:	Info

Figure 5-2 Decision Table Properties

5.2 Tuning of the synchronization

To optimize the synchronization and publishing of projects from Rule Designer to Decision Center, do the following steps:

- Ensure that Rule Designer and Decision Center have an adequate heap size. This is important because synchronization is a computation- and memory-intensive process.
- Restructure your rule project so that the rules are split among multiple top-level packages. This is important because synchronization is done at the top package level. Many rules in a single top-level package can therefore be a significant bottleneck during synchronization.

5.3 Guidelines for benchmarking an application

To avoid performance issues in your rule projects, consider the following suggestions:

- Understand, document, and communicate your performance objectives.
- Consider recording your performance measurements as you modify your XOM and rules. It is much easier to see the impact of adding a synchronized method call to the XOM, or adding a rule with a complex join, if you have a record of previous performance measurements.
- Performance test with realistic data. There is no point optimizing for a dummy data set that bears no resemblance to the expected production workload. For example, you should test with a realistic number of concurrent executions.
- Understand the frequency with which the methods in your XOM are invoked by rules, particularly if you are doing lots of inferencing using the RETE algorithm.
- Spend time optimizing your XOM where profitable.
- Spend time getting to know the features of your profiler.
- Continue the performance measurements when the application is in production because the number of rules can increase (sometimes by as much as 10% to 50% in a year).

5.4 Analyzing the rule project characteristics

Rule Designer has a tool to evaluate the complexity of the opened rule projects.

The tool is accessed by right-clicking the project and selecting **Export** → **Rule Designer** → **Rule Project Statistics**. The result is a report that details the following project statistics:

- Properties of the projects
- Business object model characteristics
- BOM2XOM mapping statistics
- Action rules statistics
- Decision tables statistics
- Rule flow statistics
- Vocabularies statistics

The main points to check are the rule flow, the decision tables, the business object model, and vocabularies. Consider the following points when you review the statistics:

- In the rule flow statistics, verify what algorithm is used by the rule set.
- In the decision tables section, check the number of rows and the complexity by number of conditions and actions.
- In the business object model section, check the complexity of the model that is manipulated by the rule project.
- In the vocabularies section, check the number of phrases that have an impact on the usability of the rules editor.

6

Decision Server Rules: Rule engine execution tuning

The rule engine processes the rules provided to it. The basic functions of the rule engine are to read its rules dynamically at run time, to reason on objects it knows, and to keep track of changes to those objects. Finally, the rule engine will invoke the execution of rules, also known as firing the rules.

The following topics are covered in this chapter:

- Academic benchmarks usage
- Rule engine execution mode choice
- Tips and tricks for business rule applications
- Rule engine performance cost breakdown

6.1 Overview of the rule engine

Rules are provided to the rule engine, which starts the processing of the rules. The rule engine evaluates the rules against the application objects and, when appropriate, executes rule actions. The rule engine can operate on ruleset parameters, local ruleset variables, and native objects inserted into the working memory (references). Rules are evaluated according to the ruleflows contained in the ruleset archive. If there is no ruleflow, all the rules in the ruleset are evaluated. Figure 6-1 shows an overview of the rule engine and the process functions.

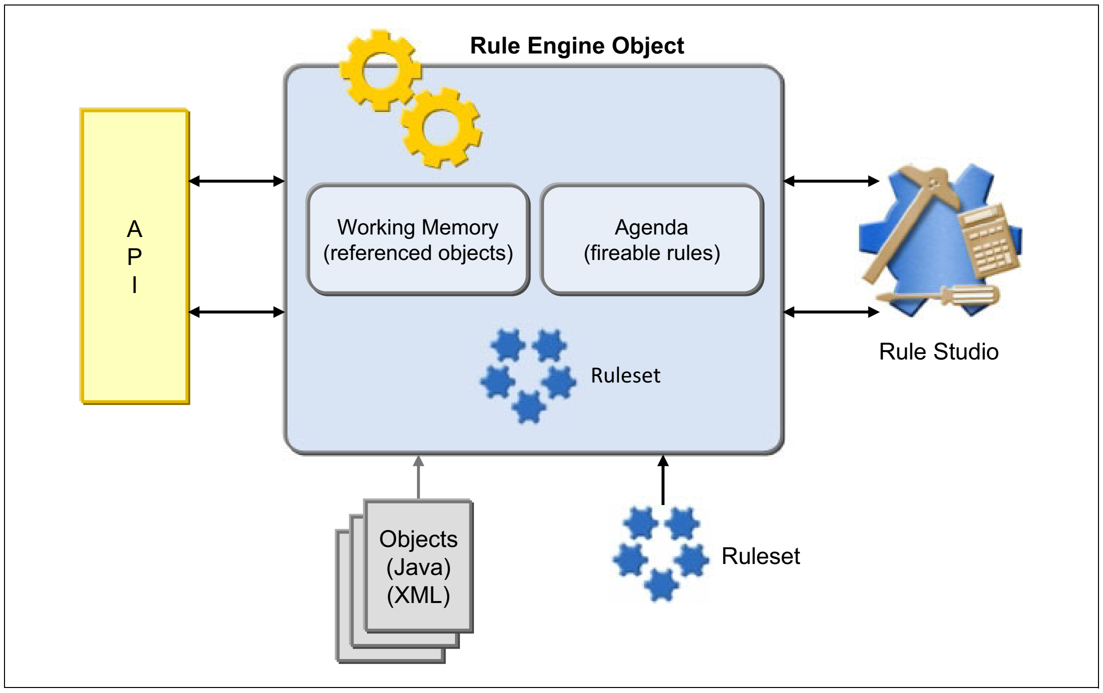


Figure 6-1 Rule engine overview

6.2 Academic benchmarks usage

A benchmark is a standard. The tendency is to look to a standard for comparison of your own results or to prove the performance quality of a product. However, benchmarks can be sterilized or static environments that have little to do with the real world environments of the industry. The following list notes some of the problems with academic benchmarks:

- Small number of rules (less than 50).
- Test worst-case RETE inference and agenda usage, which is rarely encountered with well designed applications.
- Implement solutions to combinatorial search problems, which are typically better implemented using a constraint programming (CP) engine rather than a RETE engine.
- No usage of common patterns such as ruleflow and sequential execution.
- The solutions are not verified, meaning that correctness might be sacrificed for speed. This is particularly true for Waltz and WaltzDB.

- The benchmarks are easy to manipulate because vendors can port from the original OPS5 source code into their own rule language, where they can take advantage of their product’s features.

For reference, here are descriptions of the three most famous academic benchmarks:

- Manners

  Manners is based on a depth-first search solution to the problem of finding an acceptable seating arrangement for guests at a dinner party. The seating protocol ensures that each guest is seated next to someone of the opposite sex who shares at least one hobby.

- Waltz

  Waltz was developed at Columbia University. It is an expert system designed to aid in the three-dimensional interpretation of a two-dimensional line drawing. It does so by labeling all lines in the scene based on constraint propagation. Only scenes containing junctions composed of two and three lines are permitted. The knowledge that Waltz uses is embedded in the rules. The constraint propagation consists of 17 rules that irrevocably assign labels to lines based on the labels that already exist. Additionally, there are four rules that establish the initial labels.

- Waltzdb

  Waltzdb was developed at the University of Texas at Austin. It is a more general version of the Waltz program. Waltzdb is designed so that it can be easily adapted to support junctions of four, five, or six lines. The method that is used in solving the labeling problem is a version of the algorithm described by Winston [Winston, 1984]. The key difference between the problem solving technique used in Waltz versus Waltzdb is that Waltzdb uses a database of legal line labels that are applied to the junctions in a constrained manner. In Waltz, the constraints are enforced by constant tests within the rules. The input data for Waltz is a set of lines that are defined by Cartesian coordinate pairs.

If performance is one of your major buying criteria, you are encouraged to build a proof-of-concept (POC) set of rules and data. Then, verify rule engine performance in your own environment. It is impossible to extrapolate from published academic benchmark results to your running application, with your rules and data, deployed to your OS and hardware. The POC will also allow you to evaluate the BRMS from as many angles as possible (spanning ease of use, business user accessibility, support and professional services, performance, deployment platforms, scalability, and so on).

6.3 Rule engine execution mode choice

You can choose an execution mode for each rule task in your ruleflow. By default, a rule task is assigned the RetePlus execution mode. To achieve optimal performance, you might need to choose another execution mode that is better adapted for the rules in a particular rule task.

Table 6-1 shows which mode to choose for each case.

Table 6-1 Quick view of execution mode selection

Mode	Condition
RetePlus (the default mode)	All included semantically Stateful application Rule chaining Useful in the case of many objects and limited changes
Sequential	Cover most of the customer cases Many rules, few objects, but has limitations Highly efficient in multi-thread environment
Fastpath	Rules implementing a decision structure, many objects Might have longer compilation but faster at run time Highly efficient in multi-thread environment

To determine which execution mode you should use on a rule task, analyze both the structure of the rules in the rule task, and what type of processing they perform. To make the best choice, you need to consider the following factors:

- Types of objects used by your rules
- Impact of rule actions
- Impact of tests in rule conditions
- Impact of priorities set on rules
- Who will modify the rules
- What are your performance goals

These factors are covered in more detail in the following sections.

6.3.1 Types of objects used by your rules

The preferred execution mode is based on the types of objects that are acted upon by your rules.

Working memory objects or ruleset parameters

If the objects you use in your rules are passed with ruleset parameters, we suggest that you use the Sequential or Fastpath execution modes. If they are objects that you inserted into a working memory, the RetePlus or Fastpath execution modes are preferred.

Bindings

Bindings are considered *heterogeneous* when rules do not operate on the same classes. With heterogeneous bindings, the rules might have condition parts of different heights, as shown in Table 6-2.

Table 6-2 Heterogeneous bindings

Rule	Format
Rule1	... when{A();B()} ...
Rule2	... when{A()} ...
Rule3	... when{B()} ...

If your rules define heterogeneous bindings on the Working Memory, the RetePlus or Fastpath execution modes are preferred.

If your rules define heterogeneous bindings on the ruleset parameters, all execution modes can be used, including the sequential execution mode.

Bindings are considered *homogeneous* when all of the rules operate on the same class (the same kind and number of objects), but introduce different tests. See Table 6-3.

Table 6-3 Homogeneous bindings

Rules	Format
Rule1	... when{Person(age == 12);} ...
Rule2	... when{Person(age > 20);} ...

If your rules define homogeneous bindings, all execution modes can be used, including the sequential execution mode.

6.3.2 Impact of rule actions

The choice of execution mode depends upon the effects that are generated by your rules on execution.

Modifications to the working memory

If the actions of your rules manipulate working memory objects, with the use of the IRL keywords **insert**, **retract**, or **update**, then you must use the RetePlus execution mode. Because these keywords entail modifications to the working memory, the rule engine reevaluates subsequent rules.

If you use another execution mode, the rule engine will not reevaluate subsequent rules after the modifications.

Rule chaining

When your rule actions trigger modifications in the working memory or the parameters, and when they do pattern matching on objects that have no relationship, like people and hobbies, there is chaining between your rules. Example 6-1 shows a sample rule chain.

Example 6-1 Rule chaining

```
SILVER LEVEL CUSTOMER, GREATER THAN $5000 purchase
promote to GOLD LEVEL
GOLD LEVEL CUSTOMER, GREATER THAN $5000 purchase
apply 25% discount
```

You can see in Example 6-1 that there is chaining between these two rules. They do pattern matching on two different objects (customer and purchase) and the second one will modify the level attribute of the customer object.

If you need the rules to take into account modifications done by other rules, you can use Rete execution and its rule chaining capability. Another option, if it possible to clearly split the rules that have to be executed first and then move to the other rules, you can use a sequence of rule tasks.

6.3.3 Impact of tests in rule conditions

The execution mode will differ depending on the type of conditions used in your rules.

Tests that require a working memory

If you have rule conditions that test the existence of an object or gather items in a collection directly in working memory, with the IRL keywords `exists` or `collect` and without `in` or `from` constructs, the preferred practice is to use the RetePlus or Fastpath execution modes.

Regular pattern for tests

If the tests in rule conditions have the same pattern and order, such as the tests that are generated from decision tables, the preferred practice is to use the Fastpath execution mode.

If the order of tests in rule conditions is not regular, then use the RetePlus or sequential execution modes.

6.3.4 Impact of priorities set on rules

If you set static priorities on your rules, you can use any algorithm. However, if you set dynamic priorities, that is, priorities that are defined as an expression, then you must use the RetePlus execution mode. The dynamic priorities are deprecated.

6.3.5 Impact of business user role

If business rules are modified and managed by business users, avoid using rule chaining. The performance and semantics of rule chaining is extremely sensitive to rule modification or addition. To reduce risk, use a sequence of rule tasks to simulate rule chaining.

6.3.6 Performance goals

At run time, we can divide the performance goals into three categories:

- Ruleset loading: Consider the following information if ruleset loading is a key performance goal:
  - Using RetePlus or Sequential execution.
  - Fastpath is generally slower to load.
  - The complexity of the ruleflow should be limited.
- Execution performance: Depends on the type of business rules:
  - If your rules are homogeneous (Decision table case), try Fastpath
  - If you need rule chaining:
    - Use RetePlus, but limit the use of the update flag on your BOM to accurate method or attribute. This flag forces a reevaluation of the Rete network each time this method is called.
    - Use several ruletasks in your ruleflow. With this option, you can manage a rule chaining at ruletask level (Sequential or Fastpath) and avoid the RetePlus usage.
- Concurrent executions:
  - If you know that you will have many concurrent executions, you should choose Sequential or Fastpath.

- – RetePlus does not have the scalability of Sequential or Fastpath when the number of concurrent executions is increasing.
- – The ruleflow complexity has a performance impact when the number of concurrent executions is increasing. This complexity depends on the number of subflows and on the number of ruletasks that are evaluated per execution.

Use Table 6-4 as a reference when choosing an execution mode for a rule task.

Table 6-4 Choosing an execution mode

	RetePlus	**Sequential**	**Fastpath**
Ruleset loading	Quick on small ruleset, slow on large ruleset	Quick, but slower than RetePlus on small ruleset	Slower than Sequential (rule analysis and bytecode generation)
Multi-thread scalability	Poor	Good	Good
Limitations	No	Collect in working memory. Note that not/exists are limited, no dynamic priority	No dynamic priority
Working memory objects	Yes	Yes, but only recommended with homogeneous rule conditions	Yes
Ruleset parameters	Yes	Yes	Yes
Rule chaining	Yes	No	No
Tests on existence or collection items directly in working memory	Yes	Limited	Yes
Shared test patterns	Syntactic homogeneous conditions	No (but test sharing option)	Semantic homogeneous conditions
Heterogeneous bindings	Yes	No	Yes
Dynamic priorities	Yes	No	No
Runtime rule selection that selects a few rules among many	Good	Good	Low performance (because of full evaluation cost)
Numerous rules	No	Yes	Yes

6.4 Tips and tricks for business rule applications

For many people, business rule software is a new technology. The skills required to understand the performance profile of an application using a rule engine are still developing. For example, is a database call faster or slower than invoking a set of 1000 rules? Of course, the answer depends on a number of factors. This section notes optimal practices when dealing with performance issues in business rule applications.

For applications with stringent performance targets or service level agreements, you should consider continuous performance monitoring. By investing daily in a small amount of performance monitoring, you can avoid expensive refactoring due to performance issues found at the end of your development cycle. This is particularly important for business rule applications. In these applications, the relationship between the rules being authored and the code being invoked at run time is not as obvious as with a traditional single-programming-language procedural software system.

Consider the basic sequence of operations that are required to invoke a ruleset using JRules:

1. Ruleset archive is parsed, creating an `IlrRuleset`.
2. Instantiation of an `IlrContext` for the `IlrRuleset`.
3. Creation of input data (`IN, INOUT` parameters) for the ruleset.
4. Optionally add objects to working memory.
5. Execute the ruleset.
6. Retrieve `INOUT` and `OUT` parameters.
7. Optionally, retrieve objects from working memory.
8. Reset `IlrContext` for subsequent execution.

Operation 1 is costly, and for a large ruleset containing more than 10,000 rules, it can take several minutes. However, it is a one-time cost because the resulting `IlrRuleset` should be pooled for subsequent reuse. Rule Execution Server provides out-of-the-box caching of `IlrRuleset` and `IlrContext`.

Operation 3 is often slow because this is where your code hits various expensive back-end systems, such as databases, to retrieve the data that is required by the rule engine. Caching within your code might be a useful optimization.

The time required for Operation 5 depends on the details of the rules, ruleflows, and rule tasks within your ruleset. The rules typically invoke methods in your Java Executable Object Model (XOM). It is important to determine the amount of time that is spent within your code versus the time that is spent within the rule engine or within the rules.

Additional information about optimizing execution can be found in the *Operational Decision Management information center*:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/index.jsp?topic=%2Fcom.ibm.wodm.dserver.rules.designer.run%2Foptimizing_topics%2Ftpc_optimizing_execution.html

6.5 Rule engine performance cost breakdown

Performance cost is also a part of tuning awareness. The performance cost for a JRules application might look similar to Figure 6-2 on page 73.

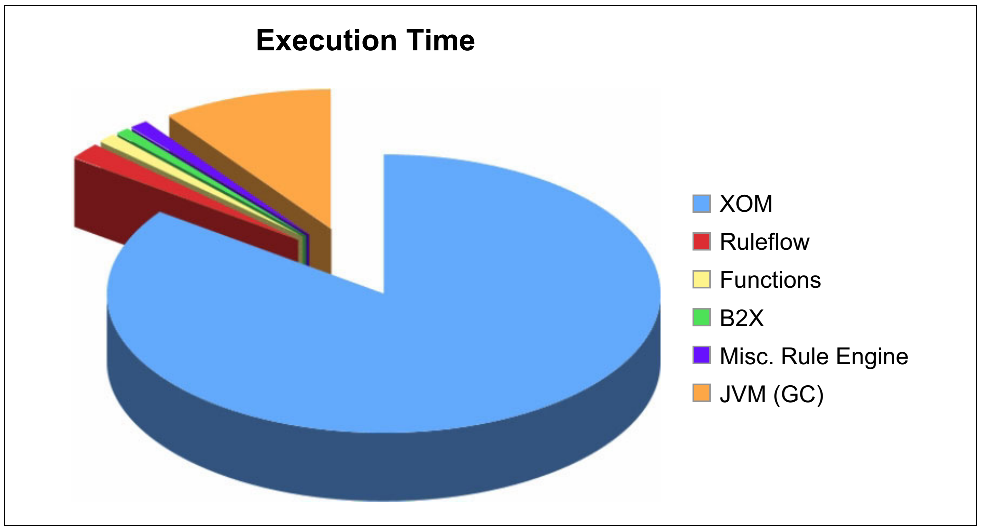


Figure 6-2 Execution Time

The majority of application time is typically spent within the XOM code invoked by the rules. It is therefore critical to get a good understanding of which methods within your XOM code are going to be invoked by rules and roughly how frequently. This might also include synchronization considerations. If you have 10 rule engines concurrently calling a synchronized method within your XOM, your throughput is going to suffer.

7

Decision Server Rules: Rule Execution Server tuning

The Rule Execution Server is a rules execution environment. Management, performance, security, and logging capabilities are provided in this environment. Rule Execution Server enables you to change business logic in a dynamic way because it lets you interact with the rule engine as a set of components. For this to work smoothly, tuning is critical. This chapter discusses the preferred tuning when the workload is limited to rules execution.

The following topics are covered in this chapter:

- Basic configuration for good performance
- Impact of ruleset characteristics on performance
- Hints
  - Impact of ruleset XOM on performance
  - Impact of execution mode on performance (trace and Decision Warehouse)
  - Architecture of your application choice (Engine, JSE, or Java EE Rule Execution Server)
  - Impact of topology on performance (JSE embedded, Java EE shared, Java EE isolated)
  - Optimization of the XU pool size and parsing
  - Rule Execution Server and clustering
- Analyzing the status of the Rule Execution Server
- Performance of Rule Execution Server with sequential algorithm
- Performance of Rule Execution Server with fastpath algorithm
- Performance of Rule Execution Server with RetePlus algorithm
- Performance impact of the number of concurrent users

7.1 Features of the Rule Execution Server

The Rule Execution Server is a complete execution and management environment for business rule applications. It provides a number of features, noted in the following list, that are required for high-performance, scalable, and manageable applications:

- Pooling of rulesets and contexts.
- Hot deployment of rulesets and asynchronous parsing.
- File and database persistence of deployed rulesets.
- Web-based system administration console.
- Runtime monitoring of execution using the Java Management Extensions (JMX) API.
- Client APIs: stateless Plain Old Java Object (POJO), stateful POJO, stateless EJB, stateful EJB, stateless JSE, stateful JSE, asynchronous execution and Message Driven EJB for asynchronous invocation, with optional execution trace or with optional auditable execution trace recording (Decision Warehouse).
- Easy service-oriented architecture (SOA) deployment: One-click configuration and integrated monitoring of rules as Transparent Decision Services.
- Simple testing and simulation of effects from rule changes (Decision Validation Services).

Figure 7-1 shows how the architecture incorporates the execution and management or monitoring stacks.

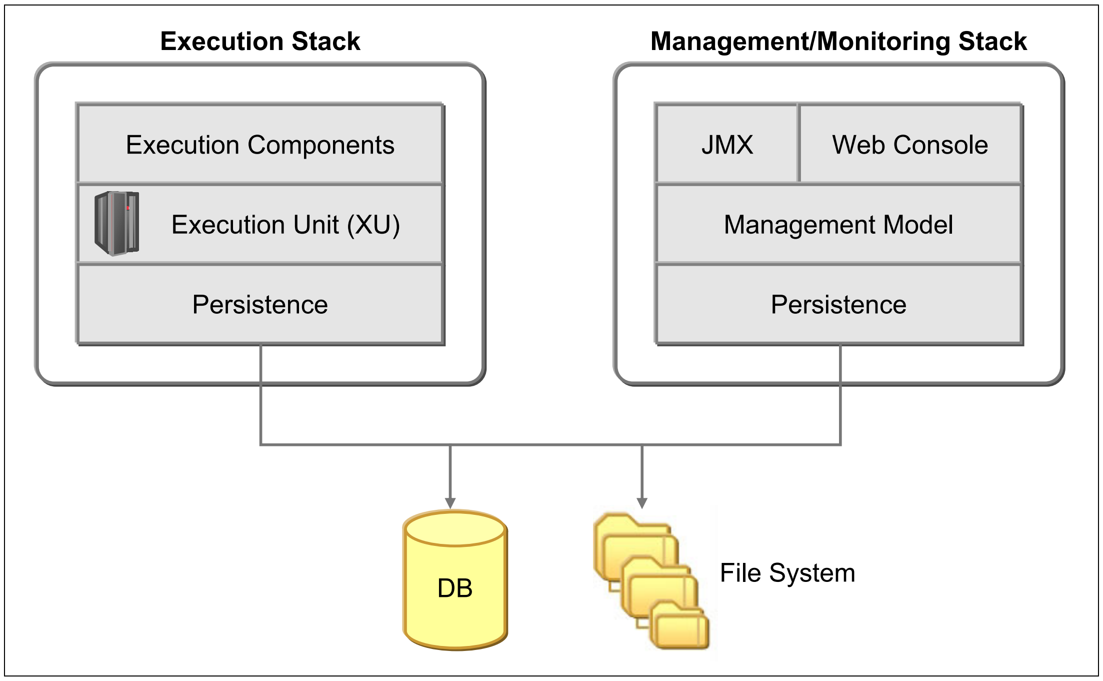


Figure 7-1 Rule Execution Server Architecture

By deploying the Rule Execution Server for Java EE or JSE applications), your business rule applications automatically take advantage of the execution and management services that are offered by Decision Server Rules. You also avoid having to implement ruleset and context pooling within your application code. Your overall application build and deployment processes can also integrate the Ant << ? >>tasks IBM provides for the Rule Execution Server.

7.2 Basic configuration for good performance

Use the following guidelines for the basic configuration settings:

- The log level in the Rule Execution Server should be set to level `Severe` or `Warning` in the production environment to increase performance. This property is accessible in the resource adapter of the Rule Execution Server or in the `ra.xml`.
- The Execution Unit (XU) pool size should be tuned to have enough connections but with a reasonable memory consumption.
- The ruleSession factory should be unique and shared between HTTP sessions, otherwise a lookup JNDI is performed for each factory creation.
- If you do not use stateful rule sessions and the transactions are at engine level, you should deactivate the transaction support in the Rule Execution Server and set `NoTransaction` instead of `LocalTransaction` in the properties of Rule Execution Server (XU) or in the `ra.xml`. If you have some code in your Executable Object Model (XOM) or rules that use a data source, the call to the data source can participate in the transaction even if the XU transaction support is disabled. The XU transaction support, set in the `ra.xml`, only concerns the engine run time itself and not other resources, even if they are called by the engine.
- Use up-to-date ruleSessions libraries:
  - Do not use old versions.
  - Integrate the correct version of the ruleSession in the classpath of your application.
- Tune the GC and memory size. On IBM JVMs, the gencon GC policy delivers good performance.
- Use the asynchronous parsing of the Rule Execution Server if you have ruleset update in the application's use cases. This option is available in the properties of Rule Execution Server (XU) or in the `ra.xml`.
- Use the execution trace and the Decision Warehouse only if it is mandatory, and tune the usage if needed (filtering).
- Limit the size of your XOM to useful classes.
- When it is turned on, Java2security will halve performance capabilities. Java2security should be turned off if possible, especially on Java EE.

7.3 Impact of ruleset characteristics on performance

The size of a ruleset has a significant impact on the following properties:

- Parsing time
- Execution with traces (number of rules, tasks, ruleflow)
- Execution with Decision Warehouse (number of rules, tasks)

Performance is greatly affected by the following issues:

- Ruleflow complexity (note that the ruleflow is interpreted, not compiled, in Decision Server Rules V8.).
- The use of dynamic filters in the rule tasks. When possible, use a fixed list of rules for a particular rule task; otherwise, the list of rules is recalculated each time that the task is executed, which has a significant affect on performance.

- Decision table complexity: Verify the size of the decision table in the Rule Studio technical view. Dividing a decision table several times can sometimes have a huge impact on the ruleset size by reducing it.
- Algorithm selection (see “Rule engine execution mode choice” on page 67).

7.4 Hints

If you encounter memory or performance issues, consider the following list of possible causes and solutions:

- Filtering what gets traced can help minimize the performance overhead that is associated with execution tracing and the decision warehouse.
- A ruleset with mixed algorithms (such as combining Rete with Sequential or Rete with Fastpath) consumes more memory than a ruleset with a single algorithm.
- You can use the ruleflow with mutual exclusive rule tasks to reduce the number of rules that are evaluated during the execution. In this way, the ruleset is segmented. Only the rules from relevant ruletasks are evaluated. For example, divide a ruleset of 20,000 rules into five ruletasks of 4000 rules. Then, create a simple condition so that only one ruletask is evaluated. The performance of this divided ruleset is similar to the performance of a ruleset with 4000 rules with a single ruletask on the same XOM.

7.4.1 Impact of ruleset XOM on performance

The XOM affects performance in the following ways:

- Execution of rules will call XOM methods, so the performance of the XOM is crucial. The Java XOM must be thread-safe to ensure system stability. A synchronized method could have a real impact on performance in case of concurrent executions. The synchronized section of your XOM should be minimized.
- The Java XOM must implement `Serializable` if you call the Rule Execution Server remotely.
- The size of the Java XOM, deployed in your EAR, has an impact at the parsing and execution level. The class loader should be the smallest loader possible. For example, do not put the complete Java EE jars in the `WEB_INF/lib` directory of your web application.
- A ruleset with an XML XOM should be configured to run in multiple simultaneous executions by configuring the pool of XML document drivers. This is done using the `ruleset.xmlDocumentDriverPool.maxSize` ruleset property. The default value is `1`.
- The **`toString()`** method performance could have a huge impact. If you are using the execution trace, with the trace filter of `setInfoBoundObjectByRule` set to `true`, the `toString()` is called for all bound objects by each executed rule.

7.4.2 Impact of execution mode on performance (trace and Decision Warehouse)

Consider the following aspects of execution mode:

- No trace:

  Execution without traces is the optimal mode in terms of memory usage and performance.
- Execution Trace:

When you choose the execution mode with trace, you will experience the following side effects:

- Memory usage increases when the execution trace is generated, the list of rules and tasks is cached, and the rule events are generated.
- On sequential or Fastpath, the ruleset property `ruleset.sequential.trace.enabled` set to `true` enables the generation of events when a rule is fired or a task is executed. The side effect is a code generation that produces event and store mapping information in the ruleset (with performance and memory impact).
- The response of the execution is bigger than the original one because the execution trace is integrated. The execution trace size depends on the ruleset characteristics (number of rules and tasks).
- The trace filtering could have a significant impact on the size of the execution trace and performance.

► Decision Warehouse execution is divided into the following actions:

- This mode executes the rule with the execution trace on; figure this into the performance cost.
- At the end of execution, a serialization of input and output parameters is done (BOM serialization).
- A complete serialization to XML of the execution trace.
- A database insertion.
- You can choose from a list of filters (as ruleset properties) what should be saved in the Decision Warehouse. To configure the execution trace, input and output parameters, and execution events (tasks and rules executed), set the properties as shown in Example 1.

Example 1   Ruleset properties

```
monitoring.filters=INFO_EXECUTION_EVENTS=true,INFO_INPUT_PARAMETERS=true,
INFO_OUTPUT_PARAMETERS= true, INFO_RULESET_PROPERTIES=false,
,INFO_INET_ADDRESS=true,INFO_EXECUTION_DURATION=true,INFO_TOTAL_RULES_FIRED=false,IN
FO_TOTAL_TASKS_EXECUTED=false, INFO_TOTAL_RULES_NOT_FIRED=false,
INFO_TOTAL_TASKS_NOT_EXECUTED=false, INFO_RULES_NOT_FIRED=false,
INFO_TASKS_NOT_EXECUTED=false, INFO_EXECUTION_DURATION=false, INFO_RULES=false,
INFO_TASKS=false, INFO_SYSTEM_PROPERTIES=false, INFO_WORKING_MEMORY=false,
INFO_BOUND_OBJECT_BY_RULE=false
```

- The Decision Warehouse is customizable. Further customization can be done through the Decision Warehouse extension point for better execution performance. You can define how data can be persisted or queried through the extension point. An asynchronous version of decision warehouse for WebSphere Application Server version 7 is available as a contribution at the following website:

  `http://www-01.ibm.com/support/docview.wss?uid=swg21433167`

Note: In the current benchmarks for the Decision Warehouse, the key bottleneck is the database insertion. This makes database tuning important if you use the Decision Warehouse without configuration changes.

7.4.3 Architecture of your application choice (Engine, JSE, or Java EE Rule Execution Server)

The choice of an architecture depends on your decision service characteristics. The following list notes the service characteristic considerations:

- Smallest memory consumption:
  - The engine alone without the Rule Execution Server
- No concurrent execution:
  - Use the engine alone
  - Use JSE Rule Execution Server
- Concurrent executions:
  - Use JSE Rule Execution Server
  - Use Java EE Rule Execution Server
- Integration with (EJB, MDB, POJO, SCA):
  - Use Java EE Rule Execution Server
- Call as a web service:
  - Use HTDS on XML XOM
  - Use MTDS on Java XOM

7.4.4 Impact of topology on performance (JSE embedded, Java EE shared, Java EE isolated)

The rule execution server can be deployed on three different topologies. Consider the following information when evaluating the impact of the topology on performance:

- Java EE Shared:
  - The XU is deployed to the application server and shared (and possibly accessed) by all of the applications that are deployed to the server.

    This is analogous to installing a device driver into an operating system, in that the application server has become globally enhanced with ruleset execution capabilities.
  - From an administrative perspective, one version of the XU is deployed. That version can be easily upgraded, started, stopped, and monitored using the application server management console or other tools.
  - The XU should be deployed on all nodes of the cluster.
  - The Java EE ruleSessions types are available.
- Java EE Isolated:
  - The XU is deployed within a single application.
  - This is a more advanced deployment option, which enables multiple applications to be deployed to the same server and allows each application to use completely separate versions of Decision Server Rules classes.
  - The XU `Jrules-res-xu-xxx.rar` is embedded in the customer application.
  - The resource is scoped to the customer application.
  - The clustering is linked to the customer application.
  - The Java EE ruleSessions types are available.

- JSE Embedded:
  - The XU is deployed as a simple Java Standard Edition JAR. Rules implement a lightweight J2C container and the Resource Adapter.
  - This mode allows the XU (and hence, the Rule Execution Server) to be used outside a Java EE application server.
  - The application server no longer manages the Resource Adapter in this mode and the Decision Server Rules pooling infrastructure is used.
  - The resource is JSE and scoped to customer application.
  - Only the JSE ruleSession is available.
- Performance impacts:
  - The performance of Java EE shared and Java EE isolated Rule Execution Server are equivalent.
  - The performance of JSE embedded is similar to Java EE shared, but the pool cannot be configured by the application server administrator.

7.4.5 Optimization of the XU pool size and parsing

The optimal XU pool size depends on the number of possible concurrent connections and the number of rulesets to execute. The limit is reached by the memory consumption. On a dual-core computer, a pool size of 50 is correct. A sizing methodology is available at the following website:

`http://www-01.ibm.com/support/docview.wss?uid=swg21400803`

The following three scenarios illustrate how the performance overhead of ruleset parsing can be minimized:

- Parsing at Rule Execution Server launch or first invocation of the ruleset. In this case, the ruleset should be parsed before the first execution. You can force it with a dummy execution or a call of the **`loadUptodateRuleset`** in IlrManagementSession API (Management Session API).
- Update of a ruleset already in use. This can be fixed by using the asynchronous parsing.
- A ruleset that is used infrequently, such as once a day, sometimes is removed from the pool running other rulesets. This can be fixed by setting `ruleset.maxIdleTime` to `0` on this ruleset. The ruleset will then stay in the pool even it is not used by a session.

See the paper *Minimizing the effect of ruleset parsing on the performance of Rule Execution Server* for details. You can access the paper at the following website:

`ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/wsw14089usen/WSW14089USEN.PDF`

7.4.6 Rule Execution Server and clustering

The Rule Execution Server supports clustering. Each XU is independent at pool level but shares the same Rule Execution Server persistence. The XUs do not share a parsed ruleset. Each time a ruleset is run, for the first time or if the ruleset is not in the pool on a node, the XU will parse the ruleset.

Cluster-wide JMX Servers (as provided by most application servers) allow the Rule Execution Server Management Console to discover and interact with multiple XUs. This occurs no matter where or how they are deployed across the cluster. Providing a single administration facility for all the rule engines deployed across a cluster of machines allows system administrators to gain an understanding of the deployment topology and diagnose runtime or configuration errors.

7.5 Analyzing the status of the Rule Execution Server

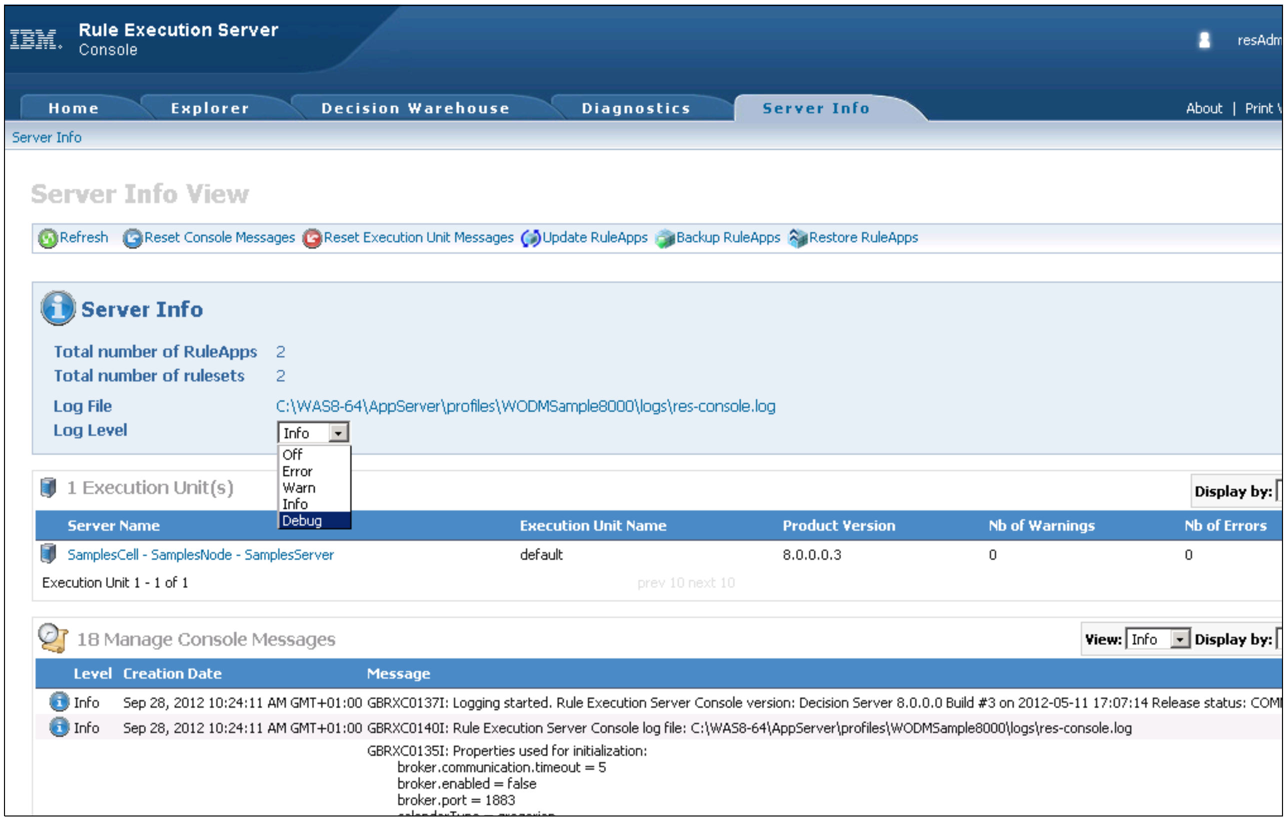
The Rule Execution Server's status is visible through a tool that is accessible using the Rule Execution Server console. The console is typically available at http://*hostName:port*/res where hostName and port are the host and HTTP port of the server.

7.5.1 Retrieving ruleset caching details from Rule Execution Server Console

You can get detailed ruleset caching information (XU dump) from the Rule Execution Server Console. This data includes how many free connections there are in the pool, which rulesets have been parsed, how much free memory is left, and so on.

To access ruleset caching information, complete the following steps:

1. Open the RES console, and click the **Server Info** tab.
2. Change the `Log Level` of the Rule Execution Server Console to `Debug` (Figure 7-2).



Total number of RuleApps 2
Total number of rulesets 2
Log File C:\WAS8-64\AppServer\profiles\WODMSample8000\logs\res-console.log
Log Level Info

Server Name	Execution Unit Name	Product Version	Nb of Warnings	Nb of Errors
SamplesCell - SamplesNode - SamplesServer	default	8.0.0.0.3	0	0

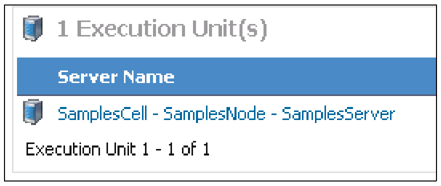
Execution Unit 1 - 1 of 1

Level	Creation Date	Message
Info	Sep 28, 2012 10:24:11 AM GMT+01:00	GBRXC0137I: Logging started. Rule Execution Server Console version: Decision Server 8.0.0.0 Build #3 on 2012-05-11 17:07:14 Release status: COM
Info	Sep 28, 2012 10:24:11 AM GMT+01:00	GBRXC0140I: Rule Execution Server Console log file: C:\WAS8-64\AppServer\profiles\WODMSample8000\logs\res-console.log

```
GBRXC0135I: Properties used for initialization:
    broker.communication.timeout = 5
    broker.enabled = false
    broker.port = 1883
```

Figure 7-2 Access to the Execution Unit status

3. Click the XU you want to investigate (still on the Server Info tab, see Figure 7-3).

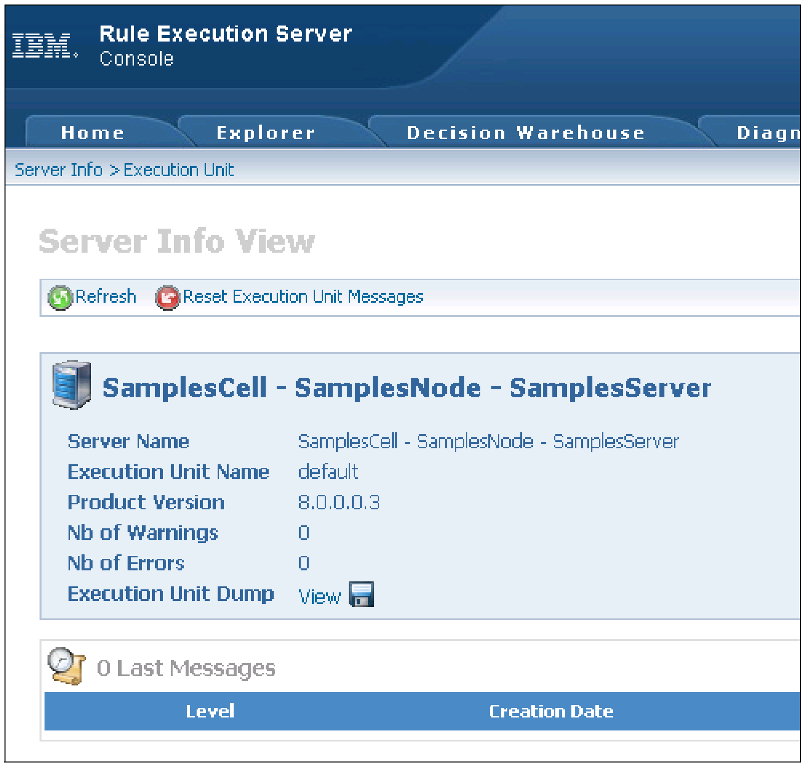

1 Execution Unit(s)

Server Name
SamplesCell - SamplesNode - SamplesServer

Execution Unit 1 - 1 of 1

Figure 7-3 Execution Unit choice

4. In the panel that opens, click **View** (Figure 7-4).



SamplesCell - SamplesNode - SamplesServer

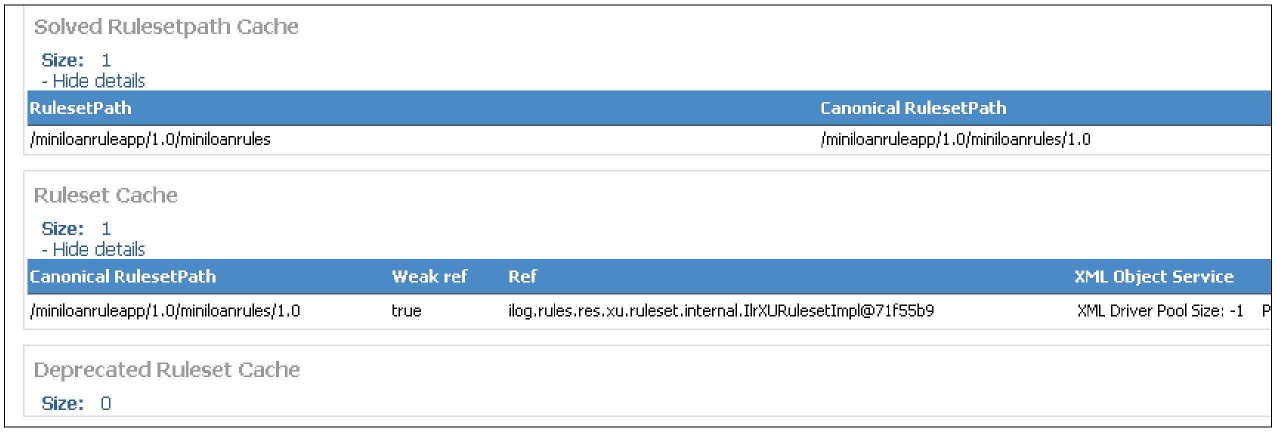
Server Name	SamplesCell - SamplesNode - SamplesServer
Execution Unit Name	default
Product Version	8.0.0.0.3
Nb of Warnings	0
Nb of Errors	0
Execution Unit Dump	View

0 Last Messages

Level	Creation Date

Figure 7-4 Access to the detailed status of Rule Execution Server

5. After the view opens, you can review the cache status. You can see which rulesets have been parsed and are stored in the cache. Figure 7-5 on page 83 shows that one ruleset is currently stored in the cache.


Solved Rulesetpath Cache

Size: 1
- Hide details

RulesetPath	Canonical RulesetPath
/miniloanruleapp/1.0/miniloanrules	/miniloanruleapp/1.0/miniloanrules/1.0

Ruleset Cache

Size: 1
- Hide details

Canonical RulesetPath	Weak ref	Ref	XML Object Service
/miniloanruleapp/1.0/miniloanrules/1.0	true	ilog.rules.res.xu.ruleset.internal.IlrXURulesetImpl@71f55b9	XML Driver Pool Size: -1 P

Deprecated Ruleset Cache

Size: 0

Figure 7-5 Status of Rule Execution caches

7.6 Performance of Rule Execution Server with sequential algorithm

We ran some tests to show the impact that a number of different factors can have on performance when you are using the sequential algorithm. For more information about selecting the appropriate algorithm, see 6.3, “Rule engine execution mode choice” on page 67. The number of rules fired is 10% of the total number of rules. The only difference is the number of rules. The tests were performed with an HTTP injector. This injector created between 20 and 45 virtual users calling the same web application to drive the ruleset through a POJO rule session or a web service call within HTDS. All benchmarks ran on IBM WebSphere Application Server V8, with Operational Decision Management 8.0 installed as a shared instance of Rule Execution Server (for example, Java EE shared topology).

7.6.1 Execution of ruleset through a POJO rule session using a Java XOM

We ran a test scenario through a POJO rule session using a Java XOM. We ran the test with three different ruleset sizes to demonstrate the impact this has. Each rule is a row of a decision table. We also ran the same tests with full execution trace enabled, and also with a limited execution trace enabled. The limited trace is composed of execution events and input/output parameters. In our tests, we obtained results showing that:

Without execution trace enabled:

- The throughput of a ruleset with 2912 rules was up to three times higher than the throughput of a ruleset with 8736 rules.
- The throughput of a ruleset with 500 rules was up to 17 times higher than the throughput of a ruleset with 8736 rules.

With limited execution trace enabled:

- The tests achieved between 65% and 90% of the throughput without execution trace, depending on ruleset size.

With full execution trace enabled:

- The tests achieved around 30% of the throughput without execution trace.

In summary:

- The number of rules has a significant effect on the throughput.
- Using filtered trace has a limited effect on the throughput.
- Using full trace has a significant effect on the throughput.

7.6.2 Execution of ruleset through a POJO rule session using an XML XOM

We ran the same tests described in 7.6.1, but using an XML XOM instead of a Java XOM. In our tests, we obtained results showing that:

Without execution trace enabled:

- The throughput of a ruleset with 2912 rules was up to three times higher than the throughput of a ruleset with 8736 rules.
- The throughput of a ruleset with 500 rules was up to 15 times higher than the throughput of a ruleset with 8736 rules.

With limited execution trace enabled:

- The tests achieved between 83% and 95% of the throughput without execution trace, depending on ruleset size.

With full execution trace enabled:

- The tests achieved around 65% of the throughput without execution trace.

In summary:

- The number of rules has a significant effect on the throughput.
- Using filtered trace has a limited effect on the throughput.
- Using full trace has a significant effect on the throughput.

7.6.3 Ruleset performance comparison of XML XOM and Java XOM

In comparing the results obtained in 7.6.1 and 7.6.2, we observed that the throughput of a ruleset using a Java XOM can be up three times higher than the throughput of an equivalent ruleset using an XML XOM.

In summary:

- In most cases, a Java XOM provides better performance than an XML XOM.

7.6.4 Execution with HTDS (web service invocation)

We ran similar scenarios to those described in 7.6.1 and 7.6.2, but using HTDS rather than using a POJO rule session. In our tests, we obtained results showing that:

Using a Java XOM:

- With a ruleset with 500 rules the throughput using a POJO rule session was up to twice the throughput of a similar ruleset invoked using HTDS.
- With a ruleset with 8736 rules the throughput using a POJO rule session was similar to the throughput of a similar ruleset invoked using HTDS.

We also compared using a Java XOM with using an XML XOM, when using HTDS. In our tests, we obtained results showing that:

- The throughput of a ruleset using a Java XOM was approximately twice the throughput of an equivalent ruleset using an XML XOM.

In summary:

- With small rulesets HTDS is slower than a POJO execution, but with larger rulesets the difference is less noticeable.
- In most cases, a Java XOM provides better performance than an XML XOM.

7.6.5 Loading time with the sequential algorithm

The first time a ruleset is executed, the ruleset is parsed. This can mean that the response time for the first execution is significantly higher than for subsequent executions. As an example, we measured the response times of the first invocations for the tests described in 7.6.1, 7.6.2 and 7.6.4. In our tests, we obtained results showing that:

- The type of XOM (Java or XML) and the use of POJO rule session or HTDS has a very small effect on the loading time.
- With a ruleset with 2912 rules, the load time is approximately half the load time of a ruleset with 8736 rules.
- With a ruleset with 500 rules, the load time is approximately 10 times faster than the load time of a ruleset with 8736 rules.

7.6.6 Conclusion

The following conclusions are based on the test results:

- The size of the rulesets has a significant impact on both the execution performance and the load time.
- Java XOM execution is faster than XML XOM.
- Usage of full execution trace has a significant performance impact, but using a filtered trace reduces the impact.
- For small rulesets, HTDS performance is slower than POJO rule session, but this is less noticeable with larger rulesets.

7.7 Performance of Rule Execution Server with fastpath algorithm

We also ran some tests to show the impact that the algorithm choice and the number of rules can have on performance. These tests were the same as described in 7.6, “Performance of Rule Execution Server with sequential algorithm” on page 84, but configured to use the fastpath algorithm, rather than sequential.

7.7.1 Performance comparison of XML XOM and Java XOM and ruleset sizes using fastpath

We ran a test scenario through both a POJO rule session and HTDS using a Java XOM and an XML XOM, all using the fastpath algorithm. We ran the test with three different ruleset sizes to demonstrate the impact this has. Each rule is a row of a decision table. In our tests, we obtained results showing that:

Using a POJO rule session:

- With a ruleset with 500 rules, the throughput using a Java XOM was approximately twice the throughput of a similar ruleset using an XML XOM.
- With a ruleset with 2912 rules, the throughput using a Java XOM was approximately 50% higher than the throughput of a similar ruleset using an XML XOM.
- With a ruleset with 8736 rules, the throughput using a Java XOM was approximately 10% higher than the throughput of a similar ruleset using an XML XOM.
- Using a Java XOM, the throughput of a ruleset with 2912 rules was twice the throughput of a ruleset with 8736 rules.
- Using a Java XOM, the throughput of a ruleset with 500 rules was seven times higher than the throughput of a ruleset with 8736 rules.

Using HTDS:

- For most ruleset sizes, the throughput using a Java XOM was approximately 20% higher than the throughput of a similar ruleset using an XML XOM.
- Using a Java XOM, the throughput of a ruleset with 500 and 2912 rules was twice the throughput of a ruleset with 8736 rules.

In summary:

- In most cases, a Java XOM provides better performance than an XML XOM.
- The number of rules has a significant impact on the throughput.

7.7.2 Performance comparison of sequential and fastpath algorithms

We compared the performance of the scenarios using sequential and fastpath algorithms. In our tests, we obtained results showing that:

Using a POJO rule session:

- With a ruleset with 500 rules, the throughput using fastpath was approximately twice the throughput of a similar ruleset using the sequential algorithm.
- With a ruleset with 2912 rules, the throughput using fastpath was approximately three times higher than the throughput of a ruleset using the sequential algorithm.
- With a ruleset with 8736 rules, the throughput using fastpath was approximately five times higher than the throughput of a ruleset using the sequential algorithm.

Using HTDS:

- With a ruleset with 500 rules, the throughput using fastpath was similar to the throughput of a similar ruleset using the sequential algorithm.
- With a ruleset with 2912 rules the throughput using fastpath was approximately four times higher than the throughput of a ruleset using the sequential algorithm.
- With a ruleset with 8736 rules, the throughput using fastpath was approximately five times higher than the throughput of a ruleset using the sequential algorithm.

In summary:

- In most cases, fastpath provides better performance than sequential. This is especially true if using decision tables (as our tests did). Tests not using decision tables might not see such a large performance difference.
- The performance difference between fastpath and sequential is particularly noticeable with larger rulesets.

7.7.3 Loading time with the fastpath algorithm

We ran similar tests to those described in 7.6.5, “Loading time with the sequential algorithm” on page 85, but using the fastpath algorithm. In our tests, we obtained results showing that:

- With a ruleset with 2912 rules, the load time is approximately 40% of the load time of a ruleset with 8736 rules.
- With a ruleset with 500 rules, the load time is approximately six times faster than the load time of a ruleset with 8736 rules.
- With a ruleset with 500 rules, the load time using sequential is approximately three times faster than the load time of a similar ruleset using fastpath.

- With a ruleset with 2912 and 8736 rules, the load time using sequential is approximately 50% faster than the load time of a similar ruleset using fastpath.

In summary:

- The loading time is longer with fastpath than with sequential.
- The delta is bigger on small rulesets than on large rulesets.

7.7.4 Conclusion

The following conclusions are based on the test results:

- The size of the rulesets has a significant impact on both the execution performance and the load time.
- Java XOM execution is faster than XML XOM.
- If using a decision table, consider the use of the fastpath algorithm. In our tests, it showed significantly improved performance over sequential.

7.8 Performance of Rule Execution Server with RetePlus algorithm

We ran some tests to show the effect that the algorithm choice and the number of rules can have on performance when using the RetePlus algorithm. These tests were the same as described in 7.6, “Performance of Rule Execution Server with sequential algorithm” on page 84, but configured to use the RetePlus algorithm, rather than sequential.

7.8.1 Performance comparison of XML XOM and Java XOM and ruleset sizes using RetePlus

We ran a test scenario using a POJO rule session using a Java XOM and an XML XOM, all using the RetePlus algorithm. We ran the test with three different ruleset sizes to demonstrate the impact this has. Each rule is a row of a decision table. In our tests, we obtained results showing that:

- For all ruleset sizes, the performance of Java XOM and XML XOM was similar.
- The throughput of a ruleset with 500 rules and one with 2912 rules was similar.
- The throughput of a ruleset with 500 and 2912 rules was three times higher than the throughput of a ruleset with 8736 rules.

7.8.2 Performance comparison of sequential and RetePlus algorithms

We compared the performance of the scenarios using sequential and RetePlus algorithms. In our tests, we obtained results showing that:

Using a POJO rule session:

- With a ruleset with 500 rules, the throughput using sequential was approximately 20 times higher than the throughput of a ruleset using the RetePlus algorithm.
- With a ruleset with 2912 and 8736 rules, the throughput using sequential was approximately four times higher than the throughput of a ruleset using the RetePlus algorithm.

- RetePlus does not scale well when there are concurrent executions of a ruleset.

7.8.3 Loading time with the RetePlus algorithm

We ran similar tests to those described in 7.6.5, “Loading time with the sequential algorithm” on page 85, but using the RetePlus algorithm. In our tests, we obtained results showing that:

- With a ruleset with 2912 rules, the load time is approximately 20% of the load time of a ruleset with 8736 rules.
- With a ruleset with 500 rules, the load time is approximately 10% of the load time of a ruleset with 8736 rules.
- With all ruleset sizes, the load time using sequential is approximately half the load time of a similar ruleset using RetePlus.

7.8.4 Conclusion

The following conclusions are based on the test results, and our observations:

- The size of the rulesets has a significant impact on both the execution performance and the load time.
- The RetePlus algorithm had slower load times, and much slower throughput compared to the sequential algorithm.
- RetePlus is an algorithm for stateful execution with rule chaining, otherwise you should consider sequential or Fastpath.

7.9 Performance impact of the number of concurrent users

We ran some tests with 1000 rules, with a Java XOM, with a variable number of concurrent users using the three algorithms RetePlus, sequential, and fastpath.

The tests show the scalability of Decision Server Rules when increasing the number of concurrent users.

During our tests, we measured both the response time and throughput, and noted the following observations:

- With sequential and fastpath, the average response time stays relatively stable when the number of concurrent users is smaller than the number of cores.
- With sequential and fastpath, when the number of concurrent users is higher than the number of cores, the response time increases quickly.
- The peak throughput with fastpath was obtained with 50 users.
- The peak throughput with sequential was obtained with 30 users.
- With RetePlus, the response time increases significantly with just five concurrent executions.
- The peak throughput with RetePlus was obtained with five users.

7.9.1 Conclusion

The following conclusions are based on the test results, and our observations. The number of concurrent users has the following impacts:

- The throughput becomes stable, depending on the complexity of the ruleset, when the number of concurrent users is higher than the number of cores.
- The average response time increases with the number of concurrent users, especially when the number of concurrent users is higher than the number of cores.
- Fastpath and sequential are more scalable than RetePlus.

8

Decision Server Rules: Tuning Decision Warehouse

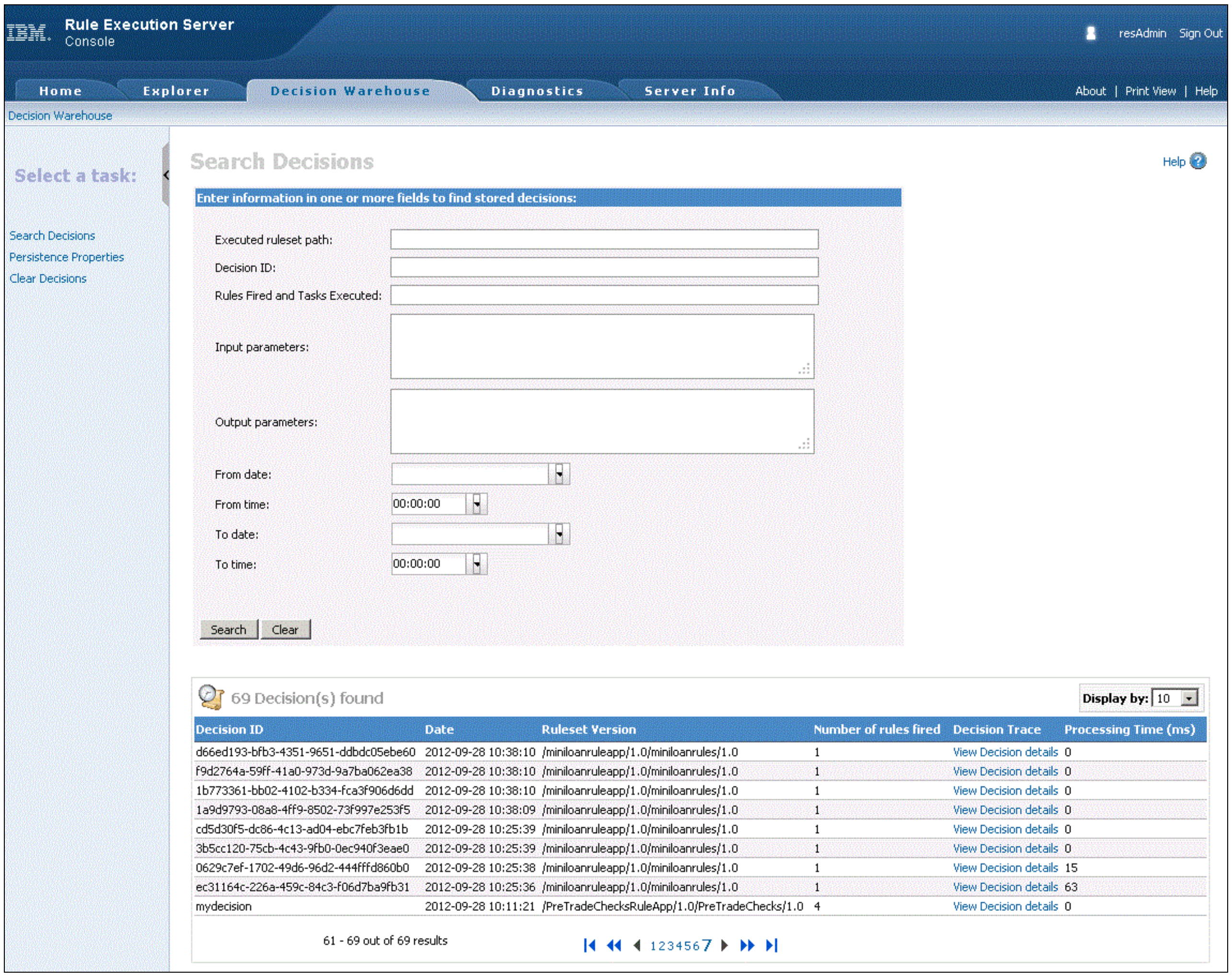
The Decision Warehouse is a module that logs decisions in production applications or in the testing through Decision Validation Services.

The following topics are covered in this chapter:

- Decision Warehouse features
- Impact of the Decision Warehouse on rule execution
- Optimization of the Decision Warehouse

8.1 Decision Warehouse interface

Figure 8-1 shows the interface used to search for and view decisions stored in the warehouse.



Enter information in one or more fields to find stored decisions:

Executed ruleset path:
Decision ID:
Rules Fired and Tasks Executed:
Input parameters:
Output parameters:
From date:
From time: 00:00:00
To date:
To time: 00:00:00

Search Clear

69 Decision(s) found — Display by: 10

Decision ID	Date	Ruleset Version	Number of rules fired	Decision Trace	Processing Time (ms)
d66ed193-bfb3-4351-9651-ddbdc05ebe60	2012-09-28 10:38:10	/miniloanruleapp/1.0/miniloanrules/1.0	1	View Decision details	0
f9d2764a-59ff-41a0-973d-9a7ba062ea38	2012-09-28 10:38:10	/miniloanruleapp/1.0/miniloanrules/1.0	1	View Decision details	0
1b773361-bb02-4102-b334-fca3f906d6dd	2012-09-28 10:38:10	/miniloanruleapp/1.0/miniloanrules/1.0	1	View Decision details	0
1a9d9793-08a8-4ff9-8502-73f997e253f5	2012-09-28 10:38:09	/miniloanruleapp/1.0/miniloanrules/1.0	1	View Decision details	0
cd5d30f5-dc86-4c13-ad04-ebc7feb3fb1b	2012-09-28 10:25:39	/miniloanruleapp/1.0/miniloanrules/1.0	1	View Decision details	0
3b5cc120-75cb-4c43-9fb0-0ec940f3eae0	2012-09-28 10:25:39	/miniloanruleapp/1.0/miniloanrules/1.0	1	View Decision details	0
0629c7ef-1702-49d6-96d2-444fffd860b0	2012-09-28 10:25:38	/miniloanruleapp/1.0/miniloanrules/1.0	1	View Decision details	15
ec31164c-226a-459c-84c3-f06d7ba9fb31	2012-09-28 10:25:36	/miniloanruleapp/1.0/miniloanrules/1.0	1	View Decision details	63
mydecision	2012-09-28 10:11:21	/PreTradeChecksRuleApp/1.0/PreTradeChecks/1.0	4	View Decision details	0

61 - 69 out of 69 results   1234567

Figure 8-1 Decision Warehouse

8.2 Decision Warehouse features

The following list identifies many of the features of Decision Warehouse:

- Logs execution trace
- Input/Output data
- Execution results
- Executed tasks
- Rules fired
- Queries
- Open API to connect third-party BI tools

Figure 8-2 shows an overview of the Decision Warehouse functions.

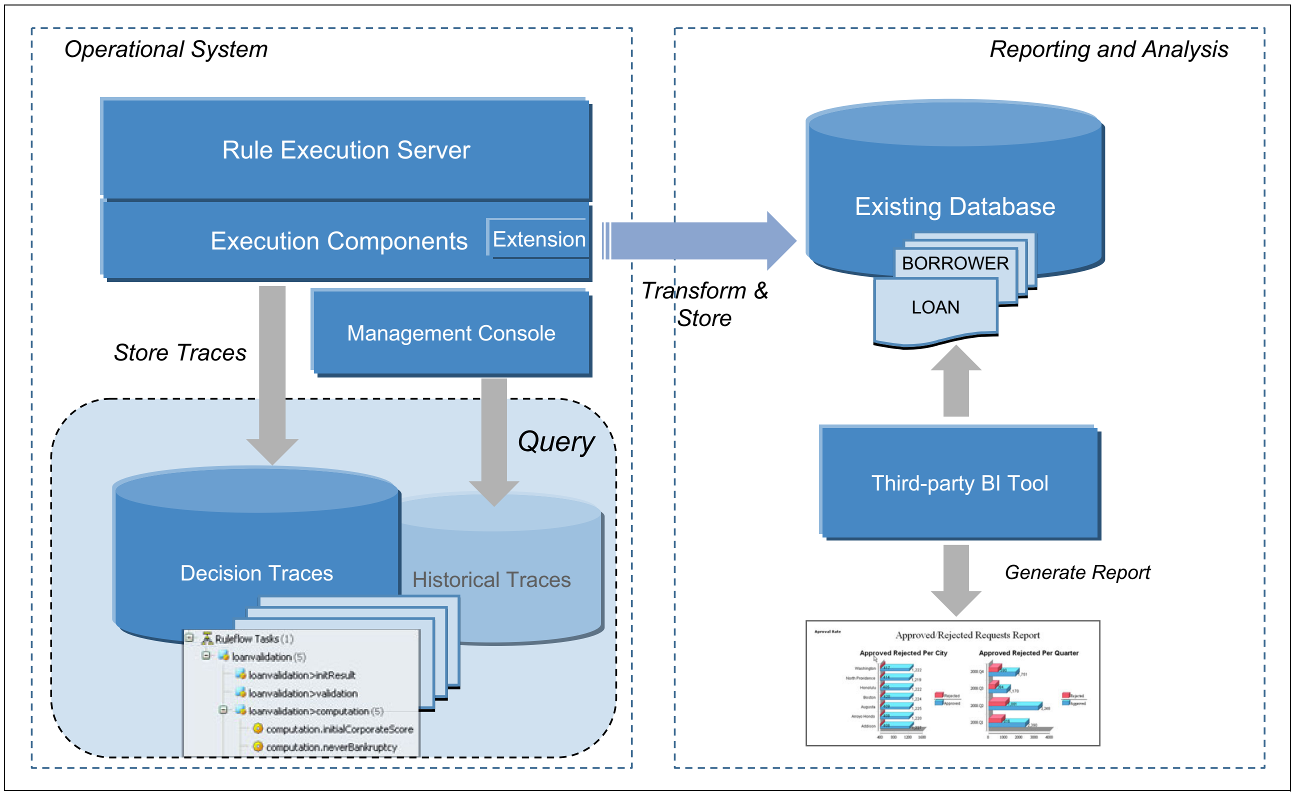


Figure 8-2 Decision Warehouse overview

8.3 Impact of the Decision Warehouse on rule execution

The following characteristics will have a significant impact on the performance of an application:

- The complexity of the object model:

  If the ruleset signature contains input/output parameters that have a large amount of data, it can take a long time to serialize and persist.

- The number of rules within the ruleset:

  Performance degrades as the number of rules fired increases.

- The level of detail you want to capture:

  The more execution details you want to capture, the longer it takes to complete the execution.

In general, the performance of the Decision Warehouse depends on the following factors:

- The performance of generating execution trace.
- The serialization performance, which divides into two principal cases for the input/output parameter serialization to BOM (depending on the size of the ruleset parameters) and the serialization of the execution trace (depending on the ruleset characteristics, such as the number of rules, the number of rules fired, and so on).
- The performance of the database server and the network used to access it.

8.4 Optimization of the Decision Warehouse

There are several ways to optimize the Decision Warehouse:

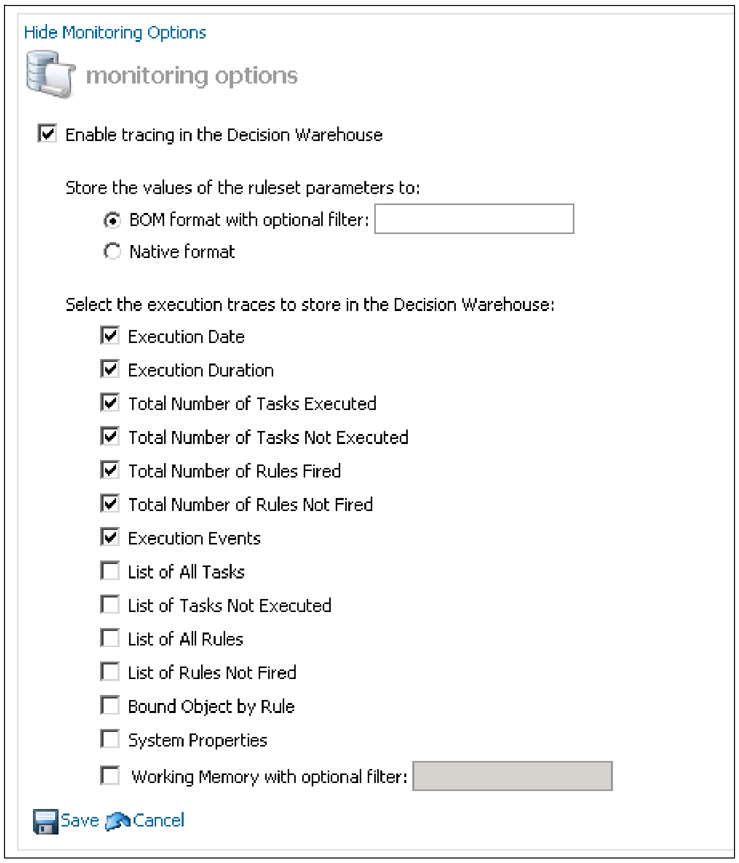
1. The Decision Warehouse is configured inside the ruleset properties. With this, you can choose the information to save. For example, there are configurable parameters to specify the ruleset properties to optimize performance. You can choose which execution information you want to capture from the following options:

   – Executed ruleset path
   – Decision ID
   – Rules fired
   – Tasks executed
   – Input/output parameters
   – Execution duration
   – Tasks not executed
   – Rules not fired
   – All the names of the rules/tasks
   – Number of rules/tasks executed
   – Custom execution output

2. You can remove the BOM serialization. See *Optimizing Decision Warehouse* at the following website:

   http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dserver.rules.res.managing/topics/tpc_res_config_optim_dw_intro.html

3. You can choose, through a list of filters (as ruleset properties), to define what is saved in the Decision Warehouse. Use the RES console, shown in Figure 8-3, to configure execution trace, input and output parameters, and execution events (tasks and rules executed).



[✓] Enable tracing in the Decision Warehouse

Store the values of the ruleset parameters to:

(●) BOM format with optional filter:
( ) Native format

Select the execution traces to store in the 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
[ ] 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 8-3 Monitoring options

The properties should be set as shown in Example 1.

Example 1 Ruleset properties

```
monitoring.filters=INFO_EXECUTION_EVENTS=true,INFO_INPUT_PARAMETERS=true,
INFO_OUTPUT_PARAMETERS= true, INFO_RULESET_PROPERTIES=false,
,INFO_INET_ADDRESS=true,INFO_EXECUTION_DURATION=true,INFO_TOTAL_RULES_FIRED=fal
se,INFO_TOTAL_TASKS_EXECUTED=false, INFO_TOTAL_RULES_NOT_FIRED=false,
INFO_TOTAL_TASKS_NOT_EXECUTED=false, INFO_RULES_NOT_FIRED=false,
INFO_TASKS_NOT_EXECUTED=false, INFO_EXECUTION_DURATION=false, INFO_RULES=false,
INFO_TASKS=false, INFO_SYSTEM_PROPERTIES=false, INFO_WORKING_MEMORY=false,
INFO_BOUND_OBJECT_BY_RULE=false,
```

4. The performance of the Decision Warehouse depends on the database. For example, the database should be optimized to handle CLOB data, not in a generic way, but for the specific case of the ruleset.
   - Further customization can be done through the Decision Warehouse extension point for better execution performance. You can define how data can be persisted or queried through the extension point. See *Decision Warehouse customization sample* at the following website:

     http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dserver.rules.samples/res_smp_topics/smp_res_dwcustom.html
   - You can also implement your own asynchronous logging of the data captured with the Decision Warehouse feature. There is a sample in the product which shows the reference architecture for using the extension point. An asynchronous version of

decision warehouse for WebSphere Application Server version 7 is available as a contribution at the following website:

`http://www-01.ibm.com/support/docview.wss?uid=swg21433167`

- There is a section in the documentation that describes the ways to optimize execution performance when using Decision Warehouse. See *Optimizing Decision Warehouse* at the following website:

  `http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dserver.rules.res.managing/topics/tpc_res_config_optim_dw_intro.html`

9

Decision Server Rules: Decision Validation Services tuning

Decision Validation Services is a module that developers use to test and simulate rules against scenarios. It gives users the ability to validate usage scenarios to ensure correctness of rulesets, or to see the potential problems. Decision Validation Services simulate the impact of business rule changes, saving you from potential critical errors before production. Tuning in this environment ensures clarity and performance in simulated situations as well as saving you time in development.

The following topics are covered in this chapter:

- Overview of Decision Validation Services
- Recommendations for DVS usage
- Recommendations to optimize big simulations

9.1 Overview of Decision Validation Services

Figure 9-1 shows an overview of the Decision Validation Services (DVS) architecture.

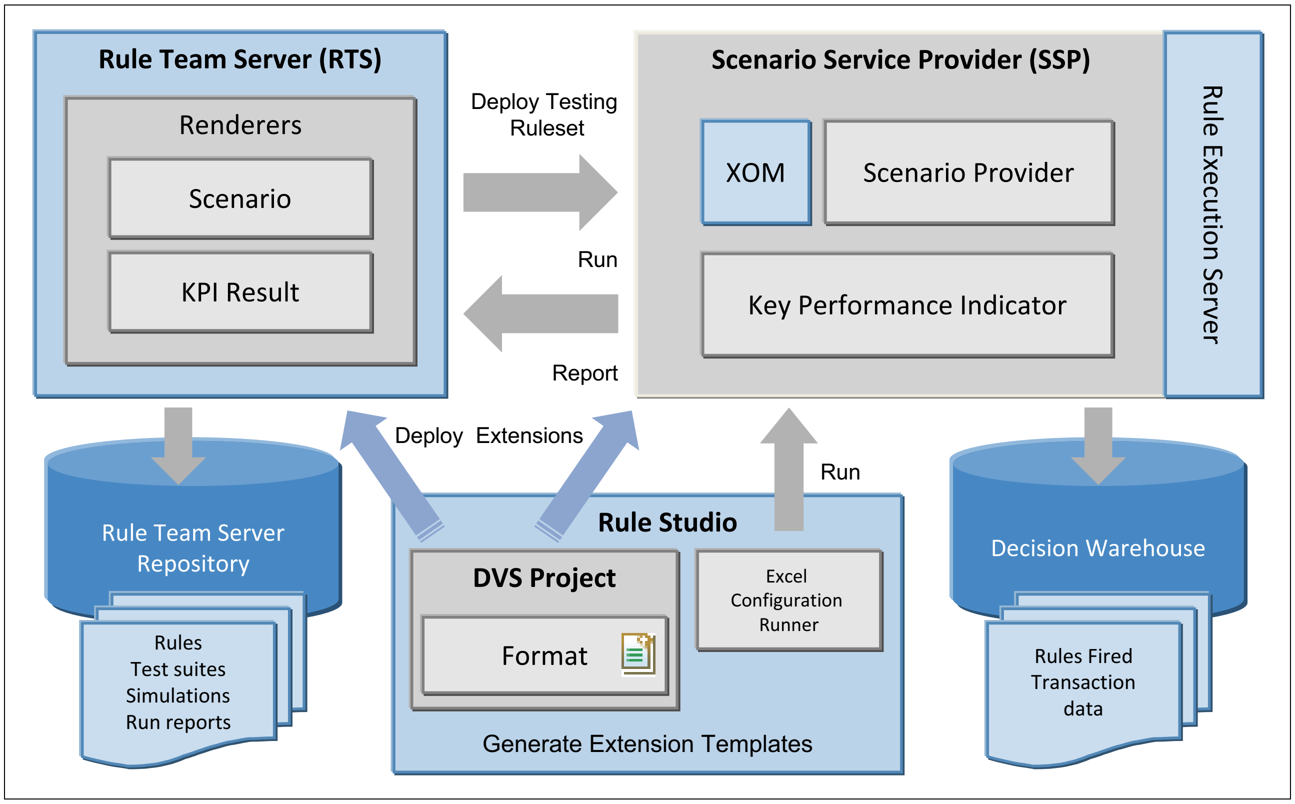


Figure 9-1 Decision Validation Services overview

The following list notes the main features of the DVS:

- Out-of-the-box ruleset testing (Decision Center)
- Business impact simulation (Decision Center)
- Scenario configuration and customization (Rule Designer)
- Audit - Decision Warehouse (Rule Execution Server)

9.2 Recommendations for DVS usage

Executing a test suite or simulation can be a long operation, depending on the number of rules and the number of scenarios to execute. The following list gives the main steps that are performed during the operation:

1. Generate a `ruleapp` archive with the set of rules to test.
2. Deploy this `ruleapp` to a Rule Execution Server console.
3. Trigger the execution of the `ruleapp` on the Scenario Service Provider, which will pass the call to the XU.
4. Obtain the report.

Most of the time consumed on Decision Center will take place during ruleset generation. For a ruleset of 10000 business rules, it takes 6 minutes for the first ruleset generation and 10 seconds for a second ruleset generation (for example, if the IRL is already cached). The rest of the time is taken in the Rule Execution Server parsing the generated archive and executing the scenarios.

9.2.1 Hints

The preferred practice is to run Rule Execution Server and Decision Center on two different machines. Using two separate machines allows Rule Execution Server to consume some resources executing a test suite or simulation, and leaves Decision Center resources (CPU and memory) unaffected.

Rule Execution Server setting recommendations

Each time a test suite execution is triggered from Decision Center, a new **`ruleapp`** is created, deployed, and parsed on the Rule Execution Server. This means that the parsing cost on Rule Execution Server cannot be avoided in the context of DVS. From the tests performed, a 10,000-rule ruleset took almost all of the 768 MB of RAM allocated for the Virtual Machine (VM). This means that without any further configuration, if a second execution launched just after the first one, it would require a new item in the XU pool. This occurrence then provokes an out of memory state for the Virtual Machine hosting the Rule Execution Server.

The following configurations are considered a preferred practice and help reduce out of memory and other performance issues:

- Dedicate a Rule Execution Server instance to DVS. Do not try to use a production Rule Execution Server for DVS tests or simulations.
- Set the size of the XU pool[1] according to the total available memory or maximum amount of memory of a ruleset. For example, if you have one GB allocated for your Virtual Machine and are executing a ruleset that takes 300 MB of memory, the size of the pool for the XU must be less than six. The size of the pool must not be under two because a test suite runs two rulesets, plus the ruleset to test, and the ruleset that contains the tests.
- Set the size of the Scenario Service Provider pool[2] to the same size as the XU pool. On WebSphere Application Server, the creation of threads is delegated to work managers; by default it is `wm/default`. When this is done, the XU pool (and thus the memory of the VM) will not be overloaded, nor will there be a bottleneck for test suite or simulation executions.
- If you have a huge volume of scenarios, the report options usage has a great impact on performance because the Decision Warehouse stores the execution results.

For more information about configuration of DVS, refer to the following website:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dserver.rules.designer.test/topics/con_configuring_ssp.html

9.2.2 Decision Warehouse optimization

Trace settings impact performance. Tuning them is an important optimization point for the overall performance of simulations and test scenarios. Their data is valuable, so you will need to consider balance in your selections. To reduce the size of Decision Warehouse trace in the Decision Validation Service, you can use the contribution available at the following website:

http://www-01.ibm.com/support/docview.wss?uid=swg21438208

[1] Set ra.xml of eXecution Unit, or configured in the JCA connection factory of the application server.

[2] For non-WebSphere Application Server platforms, see JRULES_JOBS_POOL_SIZE in the web.xml of the Scenario Service Provider WAR file.

9.3 Recommendations to optimize big simulations

Starting with Operation Decision Management version 8, the Scenario Service Provider supports a new parallel execution mode. This mode reduces the time that it takes to run large-scale simulations and to calculate the corresponding KPI values. With this new execution mode, a simulation is no longer limited to a single thread but can be executed on several threads on the same Scenario Services Provider.

Use the new parallel simulation API to access this new mode, especially at KPI level and when configuring the Scenario Service Provider pool.

For more information and a sample description of parallel execution, visit the following website:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dcenter.samples/topics/smp_tsdvs_scenarioprovider.html

An article named *Running large-scale simulations with WebSphere Operational Decision Management V 8* explains in detail the usage of this execution mode. It is available at the following website:

https://www.ibm.com/developerworks/bpm/bpmjournal/1209_brunot/1209_brunot.html

The parallel execution API is documented at the following website:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.wodm.dserver.rules.designer.test/topics/tpc_testsimulation.html

10

Decision Center tuning

Designed to enhance productivity, WebSphere ILOG Decision Center brings unsurpassed levels of usability and sophistication to business rule management. Distributed business teams can collaborate using a powerful, easy-to-use web environment to create, manage, and deploy business rules.

The following topics are covered in this chapter:

- Decision Center consoles
- Manage rules across the enterprise with confidence
- General guidelines and considerations on Decision Center
- Memory footprint of Decision Center
- CPU of Decision Center
- Tuning to maintain database performance for Decision Center
- Tuning of Decision Center

10.1 Decision Center consoles

The Decision Center has two consoles, the enterprise console and the business console. The enterprise console focuses on the needs of analysts and administrators; the business console is designed for business process experts. See Figure 10-1.

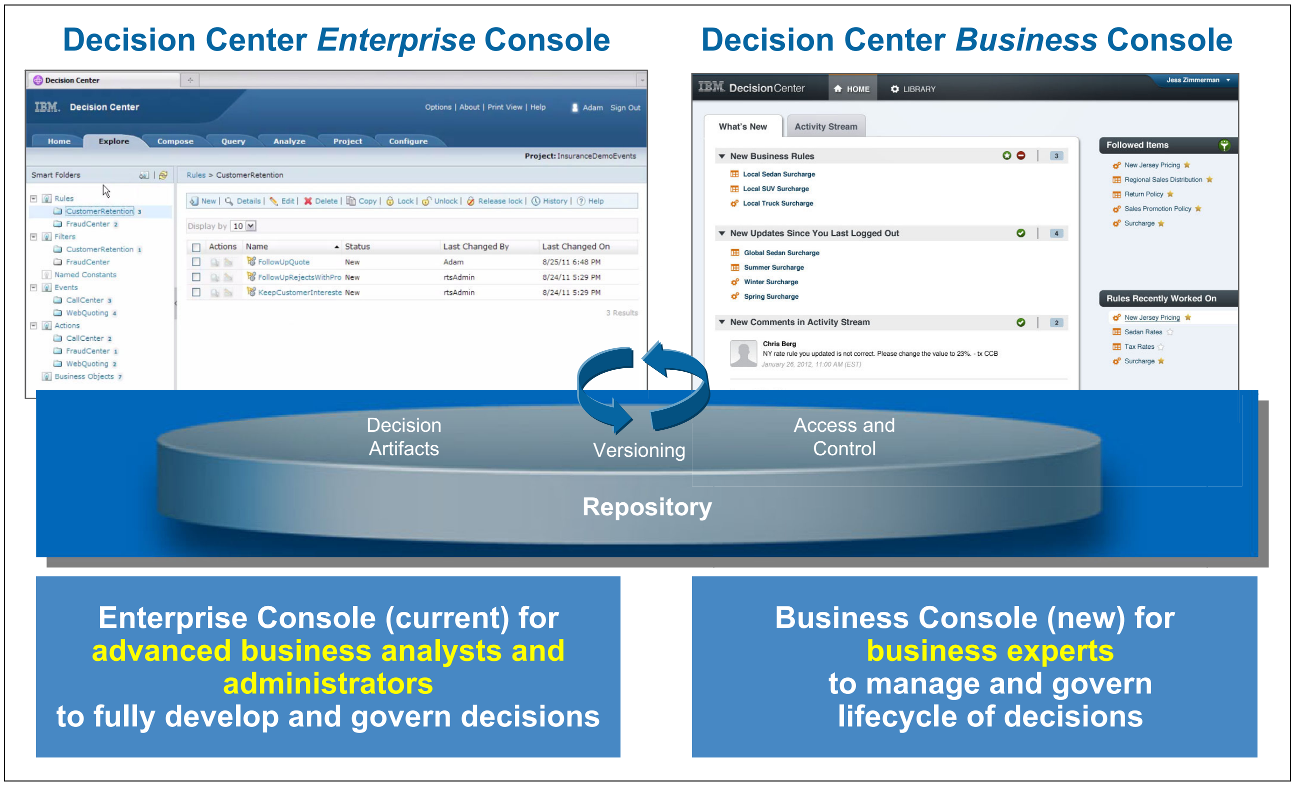

Decision Center *Enterprise* Console
Decision Center *Business* Console
Decision Artifacts
Versioning
Access and Control
Repository
Enterprise Console (current) for advanced business analysts and administrators to fully develop and govern decisions
Business Console (new) for business experts to manage and govern lifecycle of decisions

Figure 10-1 Decision Center Overview

10.2 Manage rules across the enterprise with confidence

Decision Center is easy to learn, use, and deploy. Business users can manage rules quickly and securely, saving time and money. Audit reporting and access to a central rule repository give users the peace of mind they need to focus on other tasks, keeping them ahead in a competitive business environment.

The WebSphere ILOG Decision Center gives you the ability to:

- ► Author and edit rules using a natural language and customizable business vocabulary. Rules can be expressed in graphical formats, such as decision tables and decision trees.
- ► Simplify rule projects through Smart Views (user configurable navigation panes), filters, and reports.
- ► Manage collaboration with role-based permissions, access controls, and customizable rule metadata properties.
- ► Facilitate rule maintenance with templates, point-and-click editors, error checking, and versioning.
- ► Ensure rule quality with customizable queries, rule analysis, and visual comparisons of changes between versions.

- Add powerful testing and simulation capabilities using Decision Center's integration with WebSphere ILOG Decision Validation Services.
- Extend rule management across the enterprise through Decision Center's integration with WebSphere ILOG Rule Solutions for Office.

10.3 General guidelines and considerations on Decision Center

Decision Center makes heavy use of CPU, memory, and database resources. To ensure scalability, the hardware architecture should be set up to support usage requirements. Although it is difficult to give precise recommendations due to the variety of possible configurations, this chapter should help you understand where possible bottlenecks can occur.

10.4 Memory footprint of Decision Center

Decision Center is designed to be accessed by several users at the same time. Each time a user accesses the Decision Center URL, an HTTP session is created on the server and filled with several objects. Those objects include the current IlrSession to connect to Decision Center, some Java beans to support the model of the JSF components used in the interface, and some web components (guided editor and decision table editor). The memory footprint that is required for a session is difficult to estimate because it depends upon many factors. Those factors include what actions are performed through the GUI, the virtual machine (VM), and the structure of the repository.

However, we can provide a specific example. After browsing a repository with 10,000 rules, expanding and collapsing nodes in the rule explorer, previewing some rules, editing one rule and generating a ruleset (with IRL already cached), the memory footprint of a session is around 10 MB.

10.4.1 Memory consumption and ruleset generation

During the ruleset generation, around 150 MB of short-lived objects are created. This can cause some issues if many users are performing this operation at the same time.

Ruleset generation and semantic queries are the main tasks that consume memory.

A ruleset generation is performed by any of the following uses:

- Deployment to Rule Execution Server
- RuleApp generation
- Rule Analysis features like consistency checking and completeness report
- TestSuite and simulation execution

The semantic queries are queries that contain at least one of the following sentences:

- `May apply when`
- `May become applicable when`
- `May lead to a state where`
- `Modifies the value of`
- `Uses the phrase`
- `Uses the value of`

10.4.2 Hints

If you believe the number of users connected simultaneously to a single server will not fit into the memory that is allocated for a single VM, you should deploy Decision Center to a cluster of servers and use load balancing. When deploying this way, HTTP sessions are dispatched according to the server's available memory.

Another memory-related parameter to consider is the number and size of the vocabularies that are used in a Decision Center repository. The vocabulary of a specific project is stored in memory the first time it is loaded and is shared across sessions. By default, vocabularies are stored in a pool of 10 instances. After 10 instances, other vocabularies are soft-referenced so that the VM can free the memory if needed. The same pooling algorithm applies to instances of BOMs and variable providers.

The semantic queries are slower than other queries that could be transformed as SQL queries. Do not use a semantic query to create a smart view, otherwise RTS will call this semantic query each time the **Explore** tab is clicked.

10.5 CPU of Decision Center

Depending on the number of users and which operations they frequently perform when using Decision Center, the number of CPUs can be critical. Parsing a BAL rule or a decision table is an atomic operation that takes a lock on the vocabulary instance to be used. In addition, as test results have revealed, most of the long-running operations (ruleset generation, reporting, and so on) make heavy use of the CPU. This means that if many users are accessing the same project at the same time, browsing the repository, or running long operations such as ruleset generation, the shared vocabulary might become a bottleneck. This reinforces the need to deploy Decision Center on a cluster.

10.6 Tuning to maintain database performance for Decision Center

Most Decision Center operations perform SQL queries.

Hints

Because some queries can be time consuming, it is a preferred practice to deploy the application server and the database on different machines to force the distribution of the load between Java computing and database access. This will improve the user experience when multiple users are connected to the repository.

We suggest that you set the JDBC connection pool of the data source to a number that is equivalent to the maximum number of users that will simultaneously connect to Decision Center. If the pool is too small, users might end up getting connection timeouts when their sessions are trying to access the database.

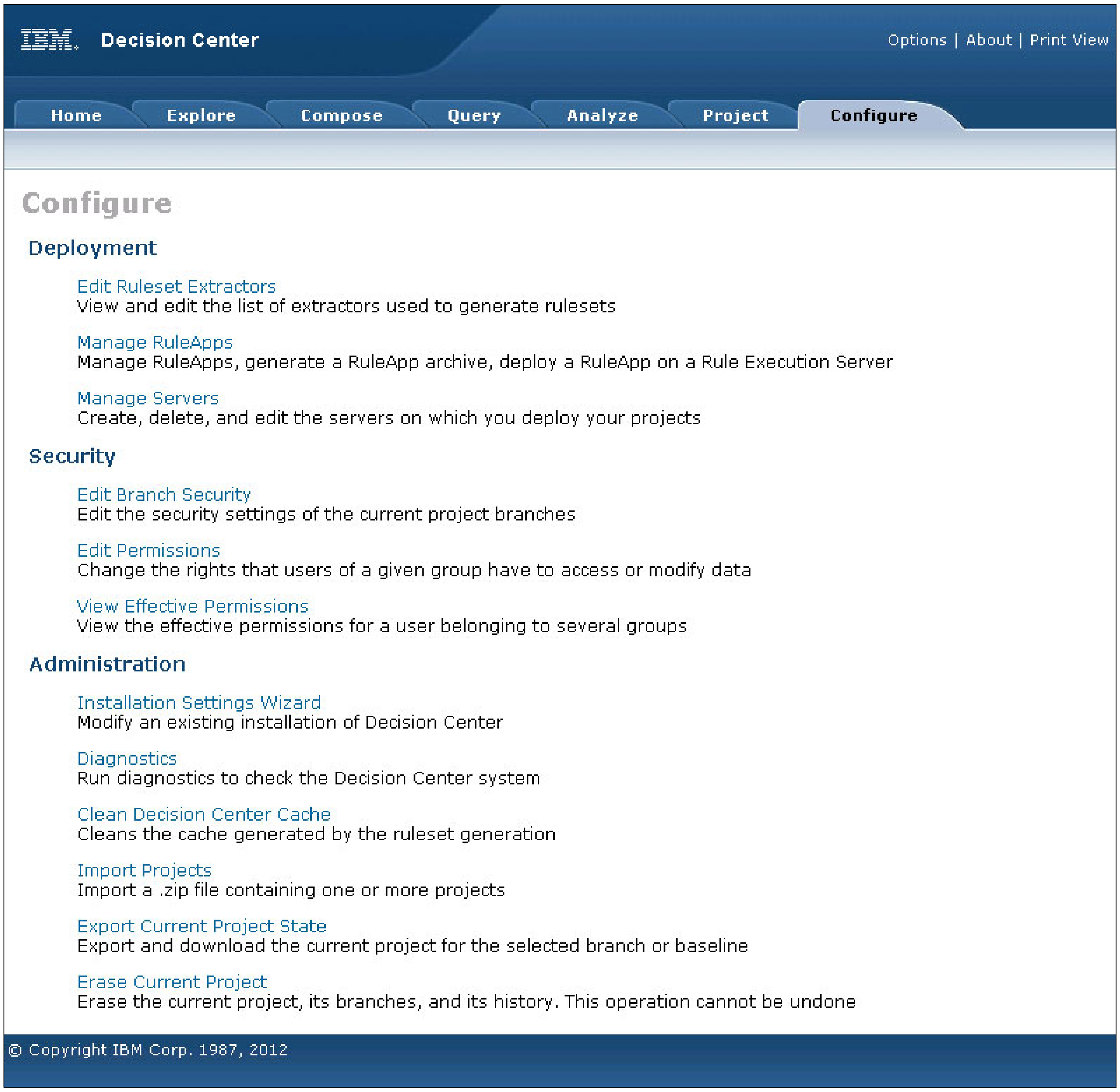
We also suggest that you set the statement cache size in WebSphere Application Server data source properties section to `50` or `100` instead of `10`. Decision Center uses a large number of different SQL requests. The performance impact is around 10% when you generate a ruleset or when you have several users using Decision Center.

10.7 Tuning of Decision Center

The tuning of Decision Center is accessible in the enterprise console of Decision Center.

10.7.1 Installation options

The installation manager of the enterprise console provides access to the main options of Decision Center (Figure 10-2).



Configure

Deployment

Edit Ruleset Extractors
View and edit the list of extractors used to generate rulesets

Manage RuleApps
Manage RuleApps, generate a RuleApp archive, deploy a RuleApp on a Rule Execution Server

Manage Servers
Create, delete, and edit the servers on which you deploy your projects

Security

Edit Branch Security
Edit the security settings of the current project branches

Edit Permissions
Change the rights that users of a given group have to access or modify data

View Effective Permissions
View the effective permissions for a user belonging to several groups

Administration

Installation Settings Wizard
Modify an existing installation of Decision Center

Diagnostics
Run diagnostics to check the Decision Center system

Clean Decision Center Cache
Cleans the cache generated by the ruleset generation

Import Projects
Import a .zip file containing one or more projects

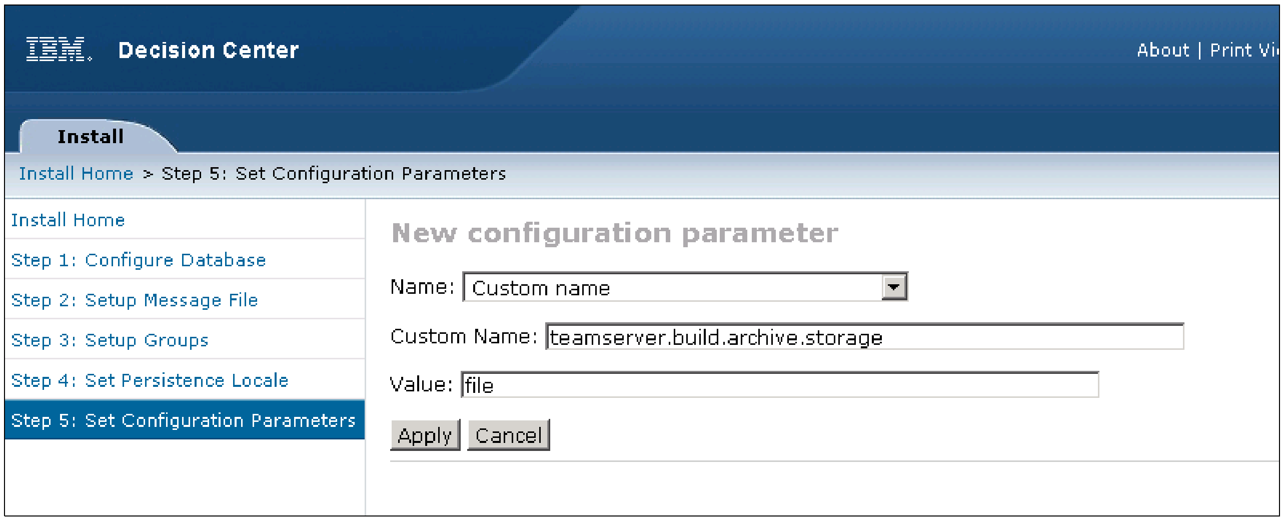
Export Current Project State
Export and download the current project for the selected branch or baseline

Erase Current Project
Erase the current project, its branches, and its history. This operation cannot be undone

© Copyright IBM Corp. 1987, 2012

Figure 10-2 Configuration of Decision Center

Modifying the build storage option limits the memory usage and stores on disk the cache of compiled business rules so this cache is persistent. To get those results, navigate to the installation manager and set the configuration parameter `teamserver.build.archive.storage` to `file` instead of memory, as shown in Figure 10-3.


IBM. Decision Center
About | Print Vi
Install
Install Home > Step 5: Set Configuration Parameters
Install Home
Step 1: Configure Database
Step 2: Setup Message File
Step 3: Setup Groups
Step 4: Set Persistence Locale
Step 5: Set Configuration Parameters
Apply
Cancel

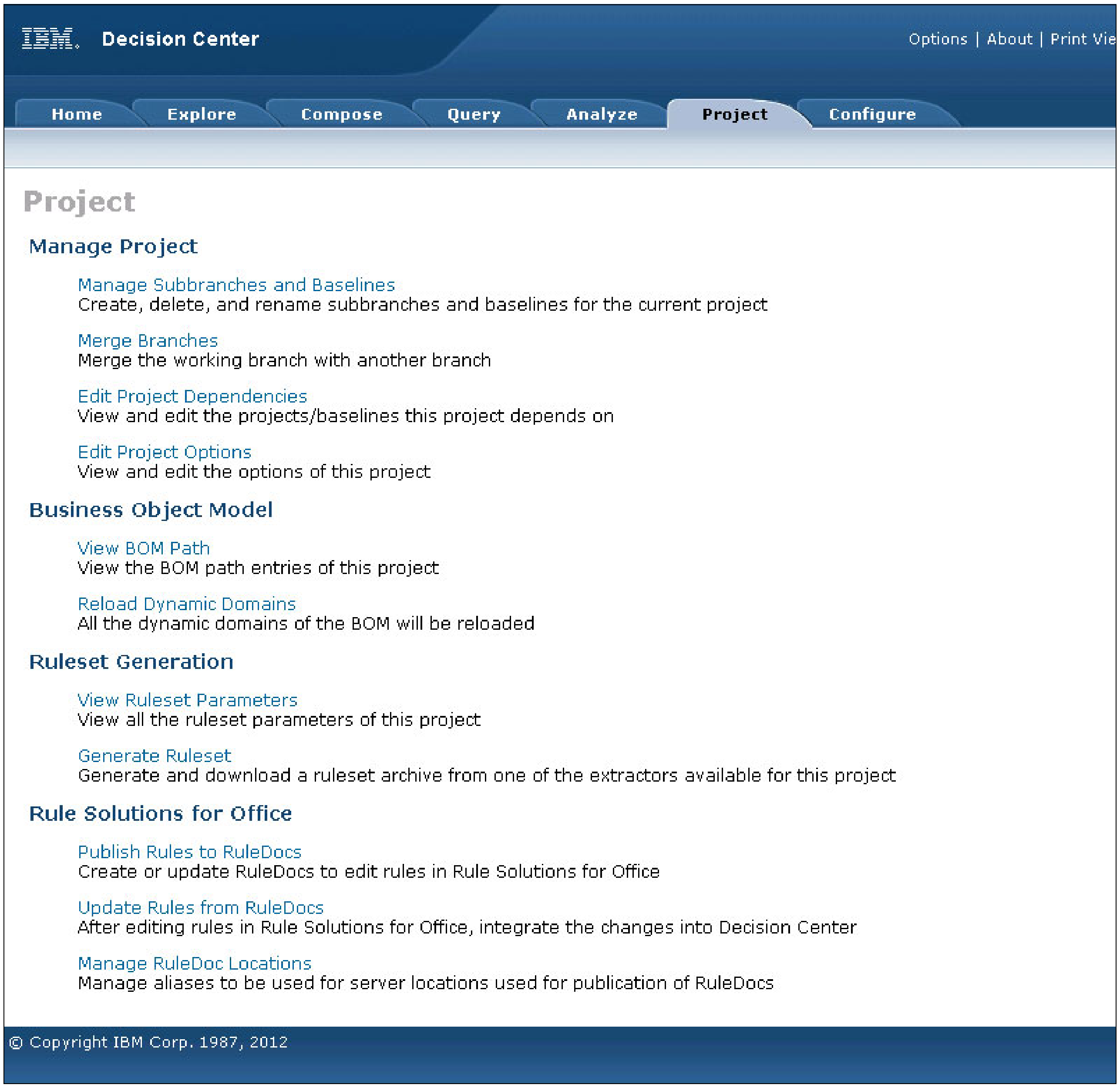
New configuration parameter

Name:	Custom name
Custom Name:	teamserver.build.archive.storage
Value:	file

Figure 10-3 Modify the build storage option

10.7.2 Project options

The options of a project are configurable in the **Project** tab of the enterprise console (Figure 10-4).



Project

Manage Project

Manage Subbranches and Baselines
Create, delete, and rename subbranches and baselines for the current project

Merge Branches
Merge the working branch with another branch

Edit Project Dependencies
View and edit the projects/baselines this project depends on

Edit Project Options
View and edit the options of this project

Business Object Model

View BOM Path
View the BOM path entries of this project

Reload Dynamic Domains
All the dynamic domains of the BOM will be reloaded

Ruleset Generation

View Ruleset Parameters
View all the ruleset parameters of this project

Generate Ruleset
Generate and download a ruleset archive from one of the extractors available for this project

Rule Solutions for Office

Publish Rules to RuleDocs
Create or update RuleDocs to edit rules in Rule Solutions for Office

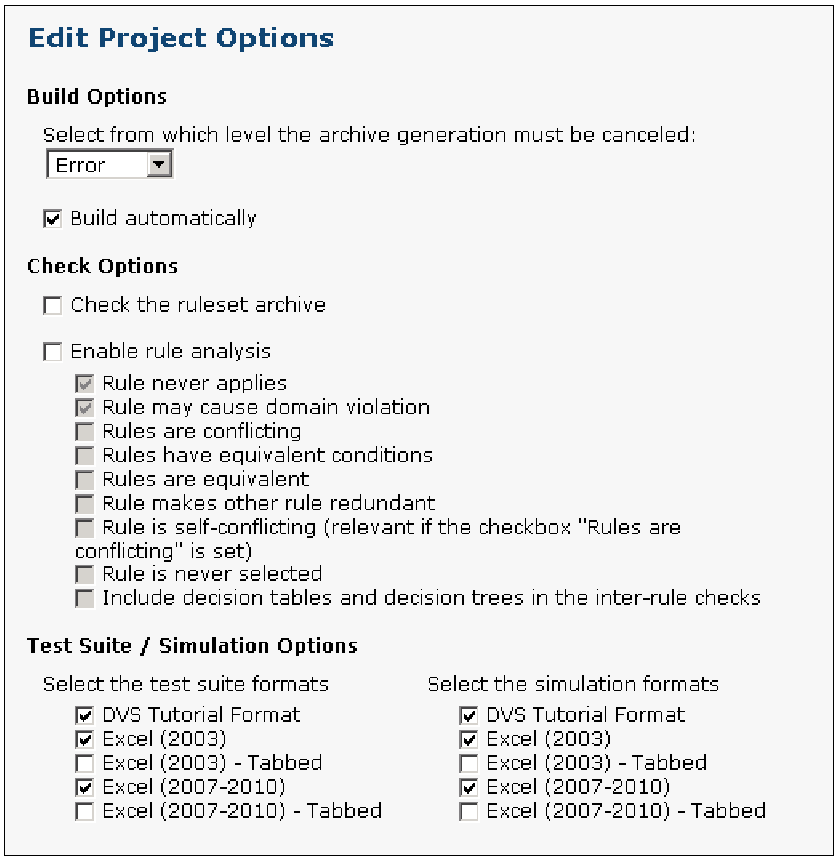
Update Rules from RuleDocs
After editing rules in Rule Solutions for Office, integrate the changes into Decision Center

Manage RuleDoc Locations
Manage aliases to be used for server locations used for publication of RuleDocs

© Copyright IBM Corp. 1987, 2012

Figure 10-4 Project configuration

The tuning of the rule project shown in Figure 10-5 limits memory consumption and CPU usage at rule edition and compilation time.


Edit Project Options

Build Options

Select from which level the archive generation must be canceled:
Error

[✓] Build automatically

Check Options

[ ] Check the ruleset archive

[ ] Enable rule analysis
- [✓] Rule never applies
- [✓] Rule may cause domain violation
- [ ] Rules are conflicting
- [ ] Rules have equivalent conditions
- [ ] Rules are equivalent
- [ ] Rule makes other rule redundant
- [ ] Rule is self-conflicting (relevant if the checkbox "Rules are conflicting" is set)
- [ ] Rule is never selected
- [ ] Include decision tables and decision trees in the inter-rule checks

Test Suite / Simulation Options

Select the test suite formats
- [✓] DVS Tutorial Format
- [✓] Excel (2003)
- [ ] Excel (2003) - Tabbed
- [✓] Excel (2007-2010)
- [ ] Excel (2007-2010) - Tabbed

Select the simulation formats
- [✓] DVS Tutorial Format
- [✓] Excel (2003)
- [ ] Excel (2003) - Tabbed
- [✓] Excel (2007-2010)
- [ ] Excel (2007-2010) - Tabbed

Figure 10-5 Project tuning

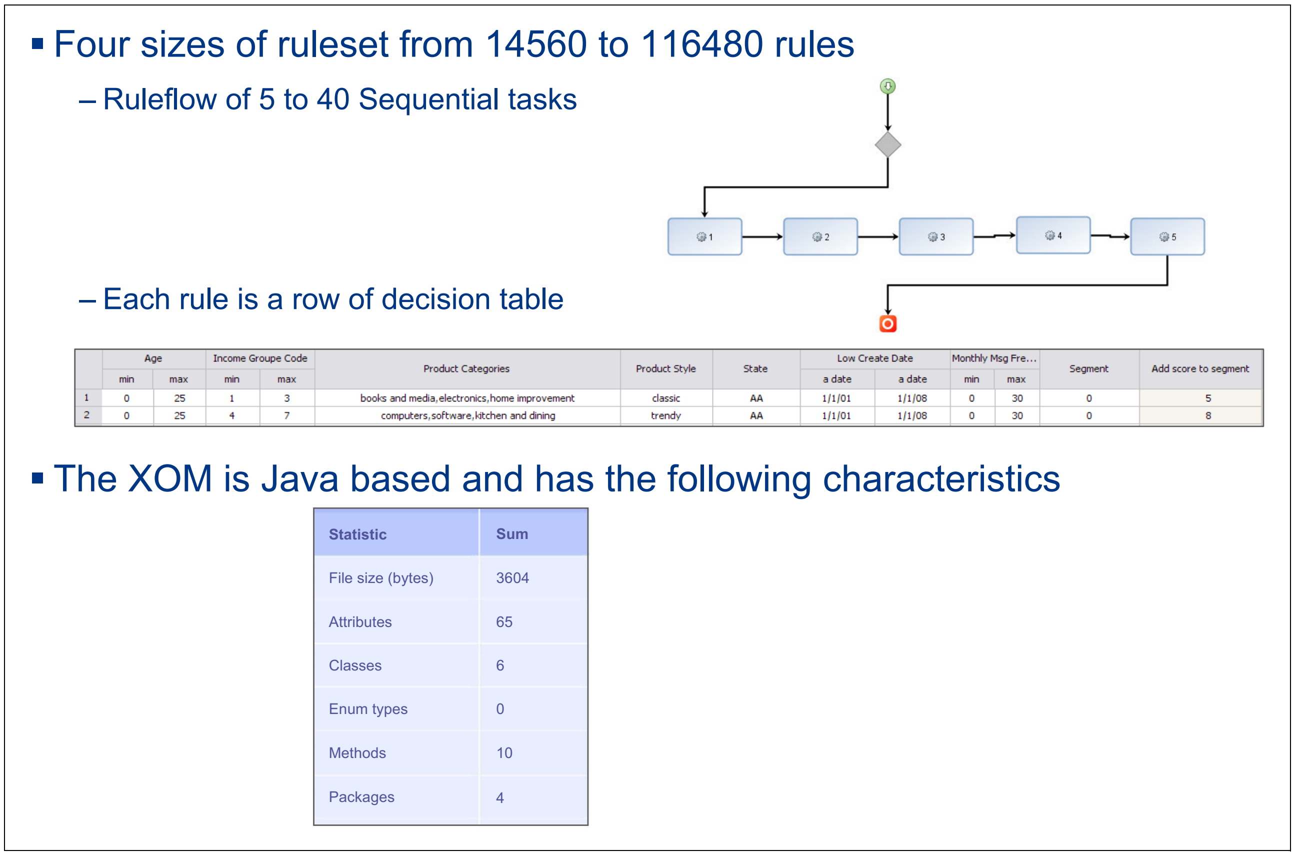
Use the following steps to tune the rule project build:

1. Disable the `Archive parsing flag` option. This can be accessed by selecting **Project** → **Edit project options** → **Check the ruleset archive**.
2. Use `Automatic build` to avoid a huge ruleset generation cost at `first ruleset generation`.
3. Disable `Rule analysis checks`.

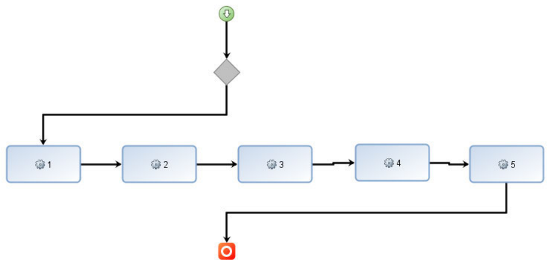
10.7.3 Performance improvements for Decision Center

The performance of Decision Center has been optimized since version 7.1, especially the building of rulesets.

To measure the improvement, we built a benchmark comparing the full build on Rule Team Server 7.1 versus Decision Center 8.0. The benchmark is built on four rule projects with different sizes. (Figure 10-6 on page 109).



- Four sizes of ruleset from 14560 to 116480 rules
  - Ruleflow of 5 to 40 Sequential tasks
  - Each rule is a row of decision table



	Age		Income Groupe Code		Product Categories	Product Style	State	Low Create Date		Monthly Msg Fre...		Segment	Add score to segment
	min	max	min	max				a date	a date	min	max		
1	0	25	1	3	books and media,electronics,home improvement	classic	AA	1/1/01	1/1/08	0	30	0	5
2	0	25	4	7	computers,software,kitchen and dining	trendy	AA	1/1/01	1/1/08	0	30	0	8

- The XOM is Java based and has the following characteristics

Statistic	Sum
File size (bytes)	3604
Attributes	65
Classes	6
Enum types	0
Methods	10
Packages	4

Figure 10-6 Rule Projects characteristics

Decision Center delivers a new ruleset build chain.

Our testing achieved performance gains from 50% to 150% versus Rule Team Server V7.1.1.1.

This optimization offers a better scalability of Decision Center as the performance gains increase with rule project size.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *Making Better Decisions Using IBM WebSphere Operational Decision Management*, REDP-4836
- *Flexible Decision Automation for Your zEnterprise with Business Rules and Events*, SG24-8014
- *Implementing an Advanced Application Using Processes, Rules, Events, and Reports*, SG24-8065
- *Proven Practices for Enhancing Performance: A Q & A for IBM WebSphere ILOG BRMS 7.1*, REDP-4775

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

Online resources

These websites are also relevant as further information sources:

- IBM WebSphere Operational Decision Managment Version 8.0 Information Center

  http://pic.dhe.ibm.com/infocenter/dmanager/v8r0/topic/com.ibm.help.doc/infocenter_homepage.html
- IBM Operational Decision Manager Version 8.0.1 information Center

  http://pic.dhe.ibm.com/infocenter/dmanager/v8r0m1/index.jsp?topic=%2Fcom.ibm.help.doc%2Finfocenter_homepage.html
- IBM Operational Decision Management home page:

  http://www-01.ibm.com/software/decision-management/operational-decision-management/
- IBM WebSphere Application Server V8.0 Information Center:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.home.doc_wasinfo_v8r0/welcome_ic_home.html
- IBM WebSphere MQ Information Center:

  http://www-01.ibm.com/software/integration/wmq/library/index.html

- Configuring and tuning WebSphere MQ for performance on Windows and UNIX

  http://www.ibm.com/developerworks/websphere/library/techarticles/0712_dunn/0712_dunn.html
- Processing messages with the WebSphere Application Server V7 and V8 WebSphere MQ messaging provider activation specifications:

  http://www.ibm.com/developerworks/websphere/library/techarticles/1110_titheridge/1110_titheridge.html?ca=drs-
- Asynchronous trace capture for Decision Warehouse

  http://www-01.ibm.com/support/docview.wss?uid=swg21433167
- Rule Execution Server engine pool sizing

  http://www-01.ibm.com/support/docview.wss?uid=swg21400803
- Reducing the size of Decision Warehouse trace when running scenario suit

  http://www-01.ibm.com/support/docview.wss?uid=swg21438208
- Running large-scale simulations with WebSphere Operational Decision Management V 8

  https://www.ibm.com/developerworks/bpm/bpmjournal/1209_brunot/1209_brunot.html
- Webphere ILOG BRMS Blog

  https://www.ibm.com/developerworks/mydeveloperworks/blogs/brms/?lang=en
- WebSphere MQ Family - Performance Reports

  https://www-304.ibm.com/support/docview.wss?uid=swg27007150
- Service Integration Bus Explorer

  https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=fe21f954-66dd-40c2-bfc4-45eb0b7d51eb
- Minimizing the effect of ruleset parsing on the performance of Rule Execution Server :

  ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/wsw14089usen/WSW14089USEN.PDF

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Operational Decision Management V8.0 Performance Tuning Guide

Redpaper™

Design choices for events and business rule applications

Runtime tuning for events and rule applications

Tuning for Decision Center

IBM Operational Decision Management (ODM) is a family of products used by IT and business users to create and manage business decision logic throughout their organization. This IBM Redpaper publication offers advice on all aspects of performance, including hardware, architecture, authoring, quality of service, monitoring, and tuning. The advice is based upon preferred practices and experience gained from real customer situations.

The book is aimed at a wide ODM audience, including IBM employees and customers, and provides useful information to new and experienced users.

Although the product family is known as IBM WebSphere Operational Decision Management (WODM), at V8.0, with V8.0.1 the the name is now simply IBM Operational Decision Manager (ODM). The performance information in this paper is based on V8.0 of this product family and differences introduced with V8.0.1 are pointed out.

REDP-4899-00