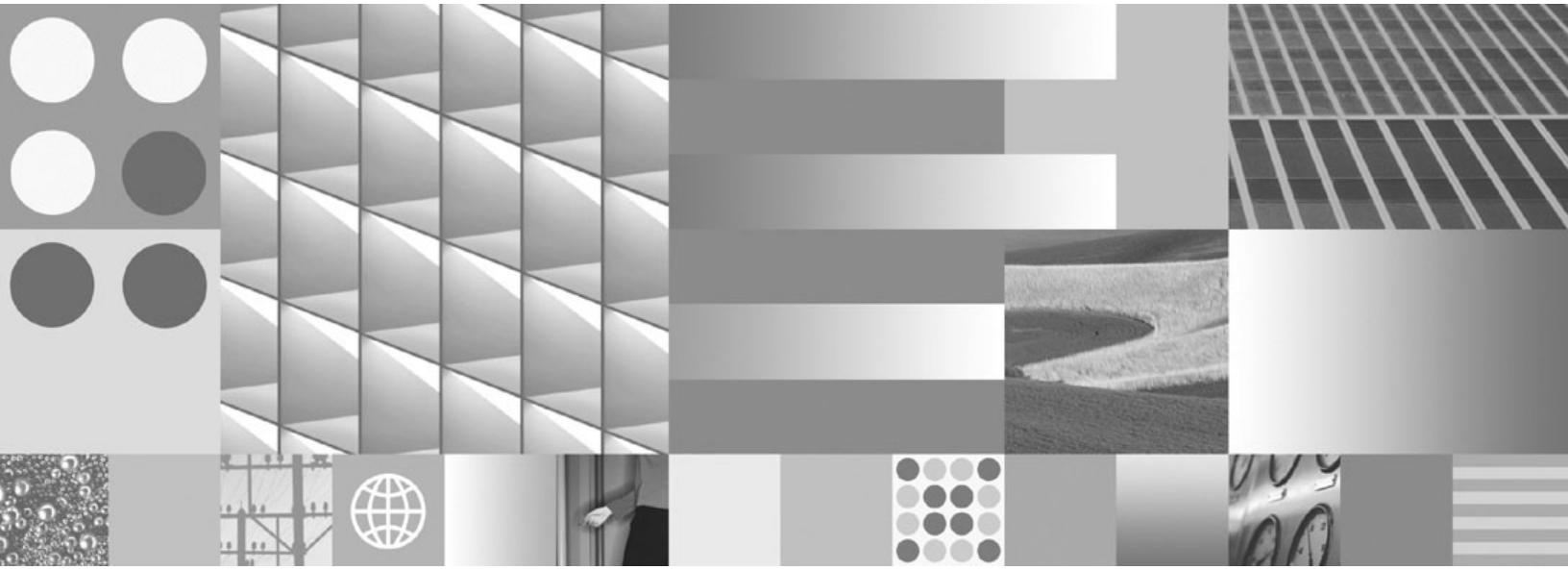
Active Correlation Technology

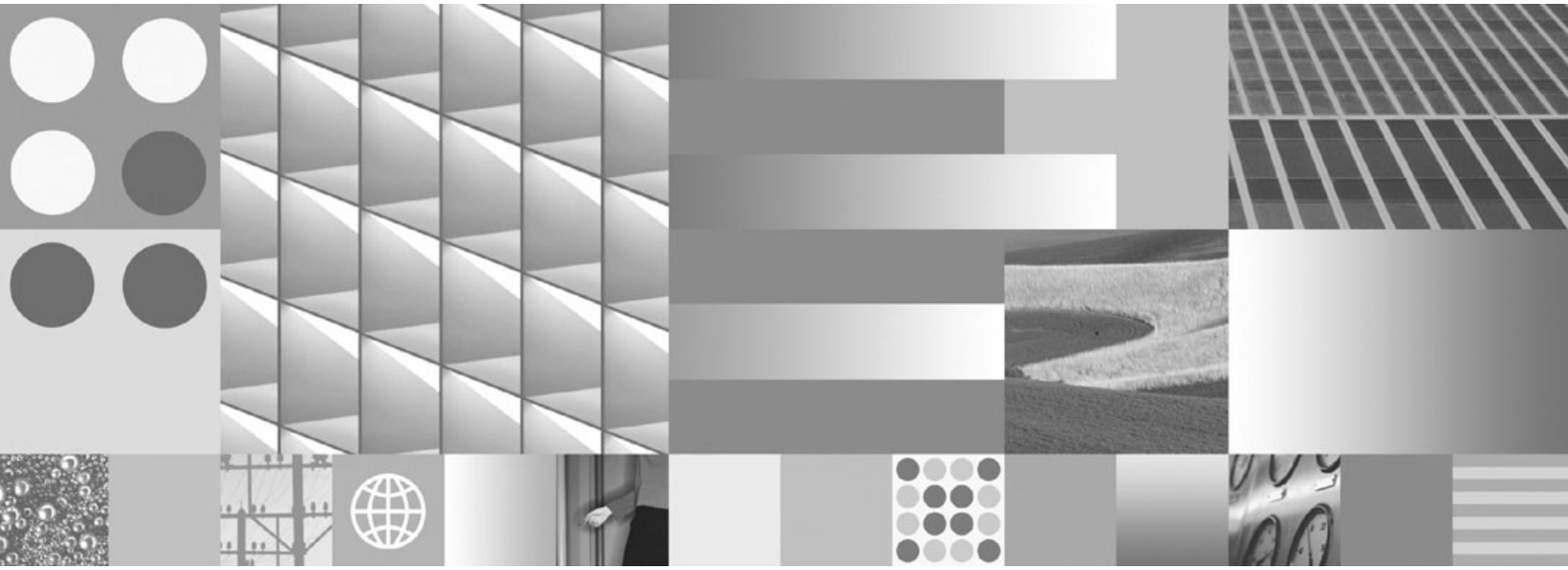
Version 6.0



Rule Writer's Guide and Reference

Active Correlation Technology

Version 6.0



Rule Writer's Guide and Reference

Note
Before using this information and the product it supports, read the information in "Notices," on page 119.

First Edition (June 2006)

About this information

This information gives an overview of the IBM® Active Correlation Technology and its role in complex event processing. The Active Correlation Technology rule language is an XML-based language for writing rules to correlate events. This information is designed for rule writers who must understand how to write rules to correlate events for their business organizations.

Contents

Part 1. Rule Writer's Guide

Chapter 1. Introduction

This introduction briefly describes complex event processing (also known as CEP) and gives an overview of the Active Correlation Technology and its role in complex event processing.

The business environment of today

Today, commercial and government organizations are dependent on electronic information processing through computer networks, and especially through the Internet. With additional technologies such as grid computing, organizations run mission-critical applications at any time and place in the world. Business processes, activity, and infrastructure, and thus our global society, are dependent on the information technology (IT) layer of organizations.

Organizations need to know what is happening with their business at all times. For example, they need to know whether mission-critical applications are available and working properly and how to detect and prevent a potential crisis in business processes, activity, or infrastructure. If a crisis occurs, they immediately need to understand the problem, how to fix it, and what caused it.

The significance of most events that pertain to business processes, activity, and infrastructure is never recognized or understood because the amount of information is too massive and too difficult to digest as it exists in individual, unrelated pieces. However, if the events are aggregated and correlated such that their relationships can be easily understood, they can yield a wealth of information.

The purpose of complex event processing is to get better information on events in real time.

Complex event processing

An *event* is simply a notification about something that has occurred.

Complex event processing is the derivation of high-level events from the analysis, correlation, and summarization of low-level events in event-driven systems. These high-level events, called complex events, are suitable for notifying people of business opportunities or problems in easy-to-understand terms or for triggering automated processes. Organizations can then operate more efficiently, with early warning of potential opportunities or problems, and with a better understanding of the root causes that change conditions in their business processes, activity, or infrastructure.

Event correlation is the process of defining and detecting patterns in event streams in real time and of implementing actions in response to related events. It is used to identify a problem based on its detected symptoms. Events can be correlated by cause, by time, by membership, or by combinations of these. Event correlation is an integral part of complex event processing.

The Active Correlation Technology

The Active Correlation Technology uses rules to detect patterns in event streams in real time. This technology is based on the understanding that in many cases, response actions should not be triggered by a single low-level event, but rather by a complex composition of events that are happening at different times and within different contexts. The Active Correlation Technology makes use of the relationships among events to provide awareness of business opportunities and problems. For example, based on the business awareness that is obtained through the correlation of events in real time, an organization can take the following types of actions:

- Offer discounted shipping for some or all customers during a holiday sale.
- Over the next 30 days, calculate the shipping cost based on the shipping carrier, the order amount, and the order quantity.
- Send customers who purchase goods that are worth more than USD500 between 1 July 2005 and 31 December 2005 a USD25 gift certificate.
- Notify an administrator if any order processing is not completed within 36 hours.
- Notify an administrator if more than four login attempts to the same computer are detected within 30 seconds.

The Active Correlation Technology consists of the following primary items:

Active Correlation Technology rule language
: An XML-based language for writing rules to correlate events. These rules can then be deployed to Active Correlation Technology runtime environments.

Active Correlation Technology engine
: The Active Correlation Technology component that processes events according to the output of the Active Correlation Technology compiler.

Active Correlation Technology rule builder
: A GUI for writing correlation rules in the Active Correlation Technology rule language.

An Active Correlation Technology runtime environment is an application in which the Active Correlation Technology engine is embedded.

Chapter 2. Rule language overview

This overview describes the key concepts of the Active Correlation Technology rule language.

A rule pattern is the representation of an event correlation situation (such as a threshold condition or duplicate event detection). The Active Correlation Technology rule language includes seven rule patterns that have been proven to represent most of the event correlation situations that IBM customers need to address. Six of the seven rule patterns define stateful rules, and one of the patterns defines a stateless rule.

Stateful rules correlate multiple events that occur during a specific time period and generate a response to those events. Stateless rules process only a single event that meets a certain condition and generate a response to that event.

stateful rule
A rule that retains state information, which is information about the characteristics of a rule instance, for the purpose of acting on a collection of events over a period of time. Rules that are defined by any of the following rule patterns are stateful rules: collection, computation, duplicate, sequence, threshold, or timer.

stateless rule
A rule that does not retain state information and therefore can act only on one event at a time. A rule that is defined by the filter pattern is a stateless rule.

Related reference

Chapter 4, "Summary of rule set organization," on page 43
This reference lists all the language elements of a rule set, a rule block, and each type of rule. It serves as a quick reference for coding a rule set.

Chapter 5, "Language element reference," on page 55
This reference describes the details of the language elements in the XML schema for the Active Correlation Technology rule language. The language elements are listed in alphabetical order, and the attributes that are available for each element are described within the topic for that element.

Chapter 6, "Glossary," on page 115
This glossary contains the terms and definitions for important concepts in the Active Correlation Technology.

Anatomy of a rule

The most basic parts of a rule are the event selection, the grouping key, the time window for stateful rules, the rule response, the activation interval, and the life cycle actions. A rule also includes expressions and variables. An expression is code that contains custom logic that can be added to a rule.

Event selection

Event selection criteria determine which events are accepted for processing by the rule. The <eventSelector> element defines the event selection criteria for a rule. Event selection applies to all rules except those that are defined by the timer

pattern. Because the timer rule does not process events, it does not contain event selection criteria.

Grouping key

Typically, each active rule has one rule instance, or copy, that is running in the Active Correlation Technology engine. However, sometimes the same rule is needed for different groups of events, which are often related to different groups of resources. The grouping key is a method for directing a rule to create a separate rule instance (or copy of itself) for each group of events that share common characteristics.

The grouping key serves as an additional form of event selection. If a rule is defined with a grouping key, and the rule receives an event with the characteristic that is defined by the grouping key, the event is sent to the rule instance that is processing the events that share that characteristic. For example, you can define a rule that collects all security events of type `Audit Failure` and define the grouping key to be the hostname attribute of an event. The rule can now be used multiple times, with a separate copy of the rule running for each unique value of the hostname attribute. You can also monitor all systems that receive the `Audit Failure` event to determine if more than 10 such events occur in a time period of 2 minutes for each host name.

The <groupingKey> element defines the grouping key for a rule, and it is valid for rules that are defined by the collection, computation, duplicate, sequence, and threshold patterns.

Time window for stateful rules

Because stateful rules correlate multiple events that occur during a specific time period, a basic part of a stateful rule is the time window, which is defined by the <timeWindow> element. The time window specifies the time period during which the stateful rule is processing to match its pattern.

Rule response

Rule response actions define the actions to take when the rule completes its processing. Each of the following language elements defines a different type of rule response action:

- <action> within <onDetection>
- <action> within <onNextEvent>
- <action> within <onTimeOut>
- <action> within <onTimeWindowComplete>

The types of rule response actions that are available to a rule are dependent on the rule pattern.

Activation interval

The activation interval defines when a rule is active and inactive. The <activationInterval> element defines the activation interval for a rule.

A rule can be activated or deactivated at a discrete point in time or by a specific event.

If you specify that a rule is to be activated, or deactivated, at a discrete point in time *and* by a specific event, the rule is activated, or deactivated, by whichever occurs first, the point in time or the receipt of the event. However, in this case, the rule might be activated or deactivated by many events throughout its life cycle. For example, a rule might be activated by an event, deactivated, activated at a defined point in time, deactivated again, and activated by another event.

The <activationByGroupingKey> element is one element that is contained within the <activationInterval> element. The <activationByGroupingKey> element contains elements that specify the events that can activate and deactivate a rule instance that is defined by the <groupingKey> element.

Life cycle actions

The life cycle actions define the actions to take at the following four primary stages in the life cycle of a rule: load, activation, deactivation, and unload.

The <lifeCycleActions> element contains the following elements that define these actions:

- <action> within <onLoad>
- <action> within <onActivation>
- <action> within <onDeactivation>
- <action> within <onUnload>

Life cycle of a rule

Each stage in the life cycle of a rule can have multiple causes and effects. By writing and including expressions within life cycle actions (as defined by the <lifeCycleActions> element), a rule writer can define the actions to take at each stage.

Stages in the life cycle of a rule

Following are the four primary stages in the life cycle of a rule:

Load The loading of the rule into the running Active Correlation Technology engine, which triggers the actions within the <onLoad> element.

Activation
The activation of the rule, which triggers the actions within the <onActivation> element.

Deactivation
The deactivation of the rule, which triggers the actions within the <onDeactivation> element.

Unload
The unloading of the rule from the running Active Correlation Technology engine, which triggers the actions within the <onUnload> element.

The activation and deactivation stages can occur multiple times in the life cycle of a rule, but the load and unload stages occur only once.

Typically, you do not need to define life cycle actions. Following are examples of when you might want to define a particular life cycle action:

- When a certain rule is loaded, you might want to create a connection to an external system (such as a database manager) that needs to be accessed within

that rule. When this same rule is unloaded, you want to drop the connection and run any necessary cleanup processes.

- When a certain rule is activated, you might want to verify that certain resources are available to that rule.
- When a threshold rule is deactivated but the threshold has not been met, and the time period is not yet over, you might want to forward a message to someone with this information.

Because the activation and deactivation of a rule can occur many times in the life cycle, any actions that you code for these stages can run frequently.

The causes and effects of each life cycle stage

Table 1 lists the causes and effects of each life cycle stage.

Table 1. Causes and effects of each life cycle stage

Life cycle stage	Causes	Effects
Load	Any of the following circumstances: • A rule or rule block is added or replaced, which causes the new rule or rules to be loaded. • The rule set is replaced in the Active Correlation Technology engine, which causes the rules in the new rule set to be loaded.	The actions within the <onLoad> element run.
Activation	The rule is activated. A rule can be activated in any of the following ways: • According to the definitions within the <activationInterval> element • Through the activate() method that is available through the act_lib variable • Through application calls to the activate() method in the Active Correlation Technology engine	If the rule is inactive, the actions within the <onActivation> element run.
Deactivation	The rule is deactivated. A rule can be deactivated in any of the following ways: • According to the definitions within the <activationInterval> element, except that the <deactivateOnEvent> element within the <activationByGroupingKey> element does not cause rule deactivation • Through the deactivate() method that is available through the act_lib variable • Through application calls to the deactivate() method in the Active Correlation Technology engine	If the rule is active, the actions within the <onDeactivation> element run.

Table 1. Causes and effects of each life cycle stage (continued)

Life cycle stage	Causes	Effects
Unload	Any of the following circumstances: • The Active Correlation Technology engine shuts down, which causes the rules to be unloaded. • A rule or rule block is removed or replaced, which causes the old rule or rules to be unloaded. • The rule set is removed from, or replaced in, the Active Correlation Technology engine, which causes the rules in the old rule set to be unloaded.	If the rule is active, the actions within the <onDeactivation> element run, followed by the actions within the <onUnload> element. Otherwise, only the actions within the <onUnload> element run.

Organization of rules

The Active Correlation Technology rule language organizes rules into rule blocks that are part of a rule set.

Rule set

The rule set contains the rules, organized into rule blocks, to be executed by an Active Correlation Technology engine. It is the rule execution unit. Each Active Correlation Technology engine acts on only one rule set at a time.

The rules that are contained in a rule set are triggered by events that are sent to the Active Correlation Technology engine. The events are delivered sequentially to the appropriate rules based on the event selection criteria for each rule, and one rule is executed at a time. The same event can apply to, and therefore trigger, multiple rules. These rules are not necessarily related, but they might be.

The order of rule blocks and rules within a rule set determines how events flow through the rule set.

Variables and imports can be defined at the rule set level for use in expressions (code that contains custom logic) throughout the scope of the rule set. An import is a programming language-specific way to access external code. A rule writer can define an import to import external modules (such as Java™ classes) for use in expressions within rules.

Rule block

The rule block is the organizational unit for grouping rules by function into domains within the rule set. A domain is the category to which a group of rules apply based on their function. For example, a domain can represent a specific geographic area, IT management discipline (such as security detection or network event correlation), or business organization (such as a specific company or division with a company).

Rule blocks can contain rules and other rule blocks. Because rule blocks can be nested, a hierarchy of rules can be constructed. For example, a rule set might contain a rule block for network event correlation, and the rule block for network event correlation might contain two other rule blocks: one for layer 2 correlation and one for IP correlation.

Therefore, a rule set provides event correlation capabilities for a variety of domains, and a rule block provides the organization for these different domains that might need access to a similar set of events.

Variables and imports can be defined at the rule block level for use in expressions throughout the scope of the rule block. The scope of a rule block includes any rules and other rule blocks that are contained in the rule block.

Rule

The rule is the correlation unit that is used to recognize relationships among events and to run the appropriate rule responses. A rule is an implementation of one of the following seven rule patterns and is organized, according to its function, into a rule block that is part of a rule set:

- collection pattern
- computation pattern
- duplicate pattern
- filter pattern
- sequence pattern
- threshold pattern
- timer pattern

Each rule can provide unique event correlation capabilities according to its pattern, and rules can be chained through event forwarding. Through this chaining of rules, the event correlation capabilities of different patterns can be combined or nested.

Variables can be defined at the rule level for use in expressions throughout the scope of the rule.

Summary

In summary, the rule set is the execution unit, the rule block is the organizational unit, and the rule is the correlation unit. A rule set contains one or more rule blocks, each of which can contain additional rule blocks. Each of the rule blocks contains rules for a specific domain. Rule blocks can be nested to construct a hierarchy of rules. The order of rule blocks and rules within a rule set determines how events flow through the rule set.

Variables and imports can be defined at the rule set or rule block level for use in expressions within rules. The scope of the variable or import is the respective rule set or rule block. Variables can also be defined at the rule level, but this limits their scope to only the rule.

Rule patterns

A rule pattern is the representation of an event correlation situation (such as a threshold condition or duplicate event detection). The Active Correlation Technology rule language defines the following rule patterns: collection, computation, duplicate, filter, sequence, threshold, and timer.

The pattern of a rule is matched when the situation that is defined by the rule occurs. When the pattern is matched, the rule concludes its processing by taking the appropriate rule response actions. While a rule is active, the rule pattern can be matched multiple times.

Rules that are defined by the filter pattern are the only stateless rules in the rule language. All other rules are stateful.

Collection pattern

A collection rule is defined by the collection pattern. It collects a group of selected events within a time interval. It is a stateful rule.

Overview

The collection pattern is used to gather similar events over a period of time. The time period is indicated by a mandatory time window, as defined by the <timeWindow> element in the rule language.

Conditions under which the rule response runs

With the collection pattern, the rule response runs when the time window is complete, as defined by the <onTimeWindowComplete> element.

Example usage of this rule pattern

An example usage of the collection pattern is a rule that does the following:

It collects events that meet the criteria of a certain event selector during the time period. When the time period ends, it summarizes the collected events into a single event that contains the total count of events and characteristic information about the summarized events.

Related reference

"Collection rule summary" on page 44
This summary lists all language elements of the collection rule.

Computation pattern

A computation rule is defined by the computation pattern. It applies a computation (through an expression) to collected events as each event is received within a time interval. It is a stateful rule.

Overview

The computation pattern runs a computation function, as defined by the <computeFunction> element in the rule language, against each event that is accepted over a period of time. The time period is indicated by a mandatory time window, as defined by the <timeWindow> element.

Conditions under which the rule response runs

With the computation pattern, the rule response runs when the time window is complete, as defined by the <onTimeWindowComplete> element. The value of the computation is available during the <onTimeWindowComplete> action.

Example usage of this rule pattern

Assume that an application is processing customer order events. An example usage of the computation pattern is a rule that does the following:

> Each time an event is received, the total value of the order is added to the total value of all orders that have occurred during the specified time period, and the updated total value of all orders is published within a user interface.

Related reference

"Computation rule summary" on page 46
This summary lists all language elements of the computation rule.

Duplicate pattern

A duplicate rule is defined by the duplicate pattern. It counts the second and subsequent events that are accepted within the specified time interval but skips the rule set processing for these events. It is a stateful rule.

Overview

The duplicate pattern is typically used to isolate similar (duplicate) events over a period of time. A duplicate event is similar in some way to a previous event, but it is not necessarily an exact copy of that event. Events are duplicates simply if they meet the event selection criteria for the rule. The time period is indicated by a mandatory time window, as defined by the <timeWindow> element in the rule language.

Conditions under which the rule response runs

With the duplicate pattern, the rule response runs at the following times:

- When the first event is detected, as defined by the <onDetection> element.
- As each duplicate event is processed, as defined by the <onNextEvent> element.
- When the time window is complete, as defined by the <onTimeWindowComplete> element.

The first event triggers the <onDetection> action even though no duplicate events have been received. The reason for this behavior is that you might want to forward the first event and skip the rule set processing for duplicate events. In this case, you can add a rule response action that forwards the first event when the <onDetection> action is triggered for the rule.

The default processing for duplicate events (the second and subsequent events) is to count a duplicate event but skip the rule set processing for a duplicate event. If you want to take additional action on a duplicate event, you can explicitly define an <onNextEvent> action. For example, in certain cases, the duplicate event represents an event that might already be logged in a database or other repository. Therefore, you might want to code an <onNextEvent> action to remove the duplicate event from these other locations.

An <onTimeWindowComplete> action can be used to create a summary record for all duplicate events that includes the number of duplicates that were processed.

Example usage of this rule pattern

Assume that a "Denial of service" message continues to occur from the same resource type (a security monitor). This indicates a possible security breach. An example usage of the duplicate pattern is a rule that does the following:

> After a "Denial of service" message occurs from the security monitor, any duplicates of that event that occur during a 30-second time period are counted but are not sent to the operator console. Also, at the end of the 30-second time period, the rule generates an event that indicates the number of "Denial of service" messages that have occurred over the time period.

Related reference

"Duplicate rule summary" on page 47
This summary lists all language elements of the duplicate rule.

Filter pattern

A filter rule is defined by the filter pattern. It takes a certain action when it accepts an event. It acts only on a single event and is therefore a stateless rule.

Overview

The filter pattern is used to act on individual events that meet the event selection criteria. Unlike the other rule patterns, it retains no associated state information (such as the history of past events).

Conditions under which the rule response runs

With the filter pattern, the rule response runs when any event that meets the event selection criteria is received, as defined by the <onDetection> element.

Example usage of this rule pattern

An example usage of the filter pattern is a rule that does the following:

> If a `ServerStatus` event indicates a `serverLoad` that is greater than 95%, the rule runs an action that pages an administrator.

Related reference

"Filter rule summary" on page 48
This summary lists all language elements of the filter rule.

Sequence pattern

A sequence rule is defined by the sequence pattern. It detects whether a certain sequence of events arrives within a time interval. The sequence can be ordered or random. A sequence rule is a stateful rule.

Overview

The sequence pattern checks for a sequence of events over a period of time, and it detects whether the sequence is complete or incomplete. An incomplete sequence is one that includes one or more of the events in the specified sequence but not all of them.

The time period is indicated by a mandatory time window, as defined by the <timeWindow> element in the rule language. Each event in the sequence is

defined by a separate <eventSelector> element within the rule. The sequence of events can be detected in either of the following orders:

- In the order in which the <eventSelector> elements are coded for the rule. In this case, when the rule detects the event that is defined by the first <eventSelector> element, the detection of the sequence is started. The rule now awaits the event that is defined by the second <eventSelector> element.
- In random order. In this case, when the rule detects any one of the events that are defined by the <eventSelector> elements, the detection of the sequence is started. The rule now awaits another one of the events that are defined by the <eventSelector> elements.

The sequence pattern is different from other rule patterns in the following primary ways:

- It has multiple <eventSelector> elements for defining which events are accepted by the rule. It requires a minimum of two <eventSelector> elements.
- When an event meets the criteria that is defined by one of the <eventSelector> elements, that <eventSelector> element is excluded from further event processing within that rule instance.
- The alias attribute on the <eventSelector> element is valid only within a sequence rule, and it uniquely names an event that is selected by a certain event selector in the sequence rule. In an expression within a filtering predicate or action, you can use the act_eventList variable to access an event in a sequence rule by its alias name.

Conditions under which the rule response runs

With the sequence pattern, the rule response runs at the following times:

- When the full sequence of events is detected within the time window, as defined by the <onDetection> element.
- When one or more events arrive but the complete sequence does not arrive within the time window, as defined by the <onTimeOut> element.

  The sequence pattern can be useful for detecting that a given sequence is incomplete. For example, if a “system down” event occurs without a subsequent “system up” event, the rule writer can code an <onTimeOut> action to handle this type of missing event.

Example usage of this rule pattern

Scenario that illustrates detection of complete sequence:
Assume that in an IT environment, an administrator wants to know whether the value of the DB2® heap size is affecting the WebSphere® Application Server, and if so, to correct this problem. Therefore, if the following events occur in the following order during a specified time period, the administrator wants to increase the value of the DB2 heap size and restart the database manager:

1. A WebSphere Application Server resource allocation exception. Assume this is an event of type `WASResourceAllocationException`.
2. The DB2 error message that states: “Not enough heap to process statement”. Assume this is an event of type `DB2NotEnoughHeap`.

For this scenario, two <eventSelector> elements are defined in a sequence rule, and the events must arrive in the order in which the <eventSelector> elements are coded (rather than in random order). The first <eventSelector> element is checking for the event `WASResourceAllocationException`, and the second <eventSelector>

element is checking for the event DB2NotEnoughHeap. Assume that the following events are presented to the system within the specified time window:

1. WASResourceAllocationException
2. DB2BackupStarted
3. WASResourceAllocationException
4. WASResourceAllocationException
5. DB2NotEnoughHeap

The rule behavior is as follows:

1. The first event, WASResourceAllocationException, is accepted. Because the criteria for the first <eventSelector> element has been met, the first <eventSelector> element is now excluded from further event processing within this rule.
2. The second event, DB2BackupStarted, is ignored.
3. The third event, WASResourceAllocationException, is ignored.
4. The fourth event, WASResourceAllocationException, is ignored.
5. The fifth event, DB2NotEnoughHeap, is accepted, and it completes the sequence. The <onDetection> rule response action runs. This action is defined to increase the value of the DB2 heap size and restart the database manager. The rule returns to its initial state.

   The first <eventSelector> element is now included in future event processing by this rule.

Scenario that illustrates detection of incomplete sequence:
Assume that a business organization wants to have all customer orders ready for delivery within 1 hour of the order request and wants to know when this does not occur.

For this scenario, two <eventSelector> elements are defined in a sequence rule, and the events must arrive in the order in which the <eventSelector> elements are coded (rather than in random order). The first <eventSelector> element is checking for the event Netsales with operationType=Order, and the second <eventSelector> element is checking for the event Netsales with operationType=Delivery. Assume that the following events are presented to the system within the specified time window of 1 hour:

1. A Netsales event with operationType=Order
2. A Netsales event with operationType=Order

The rule behavior is as follows:

1. The first event is accepted. Because the criteria for the first <eventSelector> element has been met, the first <eventSelector> element is now excluded from further event processing within this rule.
2. The second event is ignored.
3. Because a Netsales event with operationType=Delivery is not received within the specified time window, the <onTimeOut> rule response action runs. This action is defined to notify a business executive that a customer order is not ready for delivery within 1 hour of the order request. The rule returns to its initial state.

   The first <eventSelector> element is now included in future event processing by this rule.

Related concepts

"Accessing event-related information" on page 22
The following examples indicate how you can access event-related information through the variables that are provided by the Active Correlation Technology.

Related reference

"Sequence rule summary" on page 49
This summary lists all language elements of the sequence rule.

Threshold pattern

A threshold rule is defined by the threshold pattern. It collects a group of selected events within a time interval and determines, after each event is received, whether a threshold condition has been met. It is a stateful rule.

Overview

The threshold pattern collects events over a time period until a threshold value is met. The time period is indicated by a mandatory time window, as defined by the <timeWindow> element in the rule language.

The threshold pattern provides the following three options for a threshold type:

event count threshold

With this type of threshold, you can define the number of events that must meet the event selection criteria in a certain time period. The defined threshold value is compared to the number of events that have been accepted. When the event count equals the defined limit within the time window, the threshold is met.

This type of threshold can be useful for a very simple event count check. For example, it can answer the question: "Have 5 login failure events occurred within 1 minute?"

This threshold is defined by the <eventCountThreshold> element. The <eventCountThreshold> element also specifies one of the following two possible time interval modes for the time window:

fixed interval

A fixed interval begins when the first event that meets the event selection criteria is received and ends when one of the following occurs:

- The rule meets its threshold within the specified time duration.
- The specified time duration has passed.

sliding interval

A sliding interval begins when the first event that meets the event selection criteria is received. However, when the rule has not met its threshold and the specified time duration has passed, the time window adjusts (slides) the beginning time to the event reception time for a new "first" event, which is typically the next event that is accepted. The sliding interval continues to adjust in this way until one of the following occurs:

- The rule meets its threshold within the specified time duration.
- After the event that begins the time window is received, no subsequent events are received within the specified time duration.

The event that begins the time window (becomes the new "first" event) is the event with a reception time that meets this criteria:

the reception time, added to the time interval duration for the rule, is greater than the current time. Here is the criteria in the form of an equation:

event reception time + time interval duration for rule > current time

When no such event exists, the sliding interval cannot adjust the time any further, and the interval ends.

computed threshold

With this type of threshold, you can write code (or use code that was written by someone else) that performs a computation on each accepted event and returns a computed threshold value that is held in a previously defined variable. This computed threshold value is then compared to a defined threshold value to determine whether the threshold is met.

Therefore, a complex computation can be applied to create (or update) a computed threshold value, possibly using data that is saved from previous events, and the rule writer can set the defined threshold value independently of the logic that calculates the computed threshold value.

This type of threshold can be useful for the aggregation and comparison of a value to a defined threshold value. For example, it can be used to calculate the sum of the dollar amount of sales to a certain customer over a certain period of time and to compare that sum to a defined threshold value.

This threshold is defined by the <computedThreshold> element.

boolean threshold

With this type of threshold, you can write code (or use code that was written by someone else) that returns a value of `true` or `false` for each accepted event. If the value is `true`, the threshold has been met. If the value is `false`, the threshold rule continues to process until the time period is over or until it accepts another event.

This type of threshold can be useful for checking a range of values. For example, if CPU utilization must be between 30% and 80% at all times, this threshold can constantly verify that the utilization remains within that range.

This threshold is defined by the <booleanThreshold> element.

Conditions under which the rule response runs

With the threshold pattern, the rule response runs at the following times:

- When the threshold has been met, as defined by the <onDetection> element.
- When one or more events are accepted but the threshold is not met within the time window, as defined by the <onTimeOut> element.

Example usage of this rule pattern

An example usage of the threshold pattern with the event count threshold is a rule that does the following:

If more than 4 `Server unreachable` events originate from the same subnetwork within a sliding time interval of 30 seconds, the rule runs an action to check the status of a router.

Related reference

"Threshold rule summary" on page 51
This summary lists all language elements of the threshold rule.

Timer pattern

A timer rule is defined by the timer pattern. It initiates actions at regular intervals. It is a stateful rule. Although a timer rule does not process events, it can be activated or deactivated by an event.

Overview

The timer pattern is analogous to a timer that starts at the beginning of a time period and stops at the end of the time period. The time period is indicated by a mandatory time window, as defined by the <timeWindow> element in the rule language.

Unless specified not to repeat, the timer pattern repeats until the timer rule is deactivated. Therefore, when the timer rule starts, it waits the specified time period before it initiates any actions, and it repeats this behavior until either it is deactivated or the Active Correlation Technology engine shuts down.

The timer rule is unique in that it does not contain event selection criteria. The timer rule starts processing according to the activation interval for the rule, as defined by the <activationInterval> element. If the default <activationInterval> element is used and the timer pattern is set to repeat, the timer rule starts when it is loaded by the Active Correlation Technology engine and stops when the Active Correlation Technology engine shuts down. To activate a timer rule with an event, you must specify the event in the <activateOnEvent> element within the <activationInterval> element for the rule.

Conditions under which the rule response runs

With the timer pattern, the rule response runs when the time window is complete, as defined by the <onTimeWindowComplete> element.

Example usage of this rule pattern

The timer pattern can be useful for implementing cleanup rules. An example usage of the timer pattern is a rule that does the following:

> Every 30 minutes, the rule runs an action that clears harmless and informational events that have been open for longer than 48 hours.

Related reference

"Timer rule summary" on page 52
This summary lists all language elements of the timer rule.

Common and unique aspects of different rule patterns

This matrix provides a high-level overview of the common and unique aspects of the different rule patterns.

Table 2 on page 19 lists the primary language elements for the rules and shows an X in the column for each rule type under which the element is valid. The primary language elements are the direct child elements of the different rule types. The list does not include the elements that are contained within these direct child elements,

which can also vary depending on the rule type. In addition, the validity of certain element attributes can vary depending on the rule type.

Table 2. Matrix showing the common and unique aspects of different rule patterns

Element	collection	computation	duplicate	filter	sequence	threshold	timer
<comment>	X	X	X	X	X	X	X
<variable>	X	X	X	X	X	X	X
<activationInterval>	X	X	X	X	X	X	X
<lifeCycleActions>	X	X	X	X	X	X	X
<eventSelector>	X	X	X	X	X	X	
<groupingKey>	X	X	X		X	X	
<timeWindow>	X	X	X		X	X	X
<computeFunction>		X					
<booleanThreshold>						X	
<computedThreshold>						X	
<eventCountThreshold>						X	
<onDetection>			X	X	X	X	
<onNextEvent>			X				
<onTimeOut>					X	X	
<onTimeWindowComplete>	X	X	X				X

Expressions

An expression is code that contains custom logic that can be added to a rule. Expressions can also access code that is external to the Active Correlation Technology engine. In the rule language, expressions are valid only within specific contexts, or rule language elements.

Rule writers can code expressions for different purposes depending on the context and on the result that they want to obtain. Expressions are frequently used for the initialization of variables, the definition of event selection criteria, and the specification of rule response and life cycle actions.

Language elements that contain expressions

Each language element that contains an expression has an expressionLanguage attribute that identifies the programming language in which the expression is written. The Java programming language is the only supported expression language.

Expressions can be contained within the following rule language elements.

- <varInitializer> for a rule set, rule block, or rule variable
- <filteringPredicate> on <eventSelector>
- <computedValue> on <groupingKey>
- <computeFunction> on a computation rule
- <booleanThreshold> on a threshold rule
- <computedThreshold> on a threshold rule
- Rule response actions for a rule:
  - <action> within <onDetection>. This action is valid only for the duplicate, filter, sequence, and threshold rules.
  - <action> within <onNextEvent>. This action is valid only for the duplicate rule.

  - <action> within <onTimeOut>. This action is valid only for the sequence and threshold rules.
  - <action> within <onTimeWindowComplete>. This action is valid only for the collection, computation, duplicate, and timer rules.
- Life cycle actions for a rule:
  - <action> within <onLoad>
  - <action> within <onActivation>
  - <action> within <onDeactivation>
  - <action> within <onUnload>

What the Active Correlation Technology provides to help in coding expressions

To help a rule writer in coding expressions, the Active Correlation Technology provides the capability to do the following:

- Import external modules (such as Java classes) and objects for use in expressions.
- Initialize and access rule set, rule block, or rule variables.
- Through the act_event variable, access the current event that a rule is processing.
- Through the act_eventCount variable, access the number of events that have been accepted by a rule.
- Through the act_eventList variable, access the list of events that have been accepted by a rule. This includes the capability to access various attributes of an event and to access each event in a sequence rule by its alias name.
- Through the act_lib variable, access methods that include the capability to get and set variables and to control event flow through a rule set.
- Through the act_location variable, access the location, within the rule hierarchy, of an expression.
- Through the act_nodeName variable, access the fully qualified name of a node.
- Through the act_threshold variable, access the defined threshold value for a threshold rule.

Importing and accessing external modules and objects

This example indicates how you can make external code (such as Java classes) and external objects accessible to expressions. An external object is an object that an application creates to communicate with expressions.

Before you can access external code from an expression, you must make that code accessible to expressions.

An import is a programming language-specific way to make external code accessible to expressions. The <import> element contains a special type of expression that specifies the external modules (such as Java classes) to import for use in other expressions within rules. An import can be defined at the level of the rule set or a rule block.

The following <import> element contains an expression, written in the Java programming language, that imports the StaticHelper class and the Queue class, which can be referenced from other expressions:

```
<import expressionLanguage="java">
 import com.ibm.act.sample.StaticHelper;
 import com.ibm.act.test.Queue;
</import>
```

Although the use of the full class name is not required in the import statement, you should specify the full name to prevent a long compilation time. For example, the Java class should be specified as `com.ibm.act.sample.StaticHelper` rather than `com.ibm.act.sample.*` or `com.ibm.act.*`.

Accessing a static method

The following example indicates how an expression within a rule response action references the StaticHelper class after the class is imported:

```
<onDetection>
 <action expressionLanguage="java">
  StaticHelper.pageAdministrator("Too many login attempts for " + act_event.getAttribute("userID"));
 </action>
</onDetection>
```

Accessing an instance method for an object

The following example indicates how an expression within a rule response action references the Queue class after the class is imported. In this example, an external object with a name of `OutputQueueOne` and a type of `Queue` is obtained and used to put an event in a specific queue.

```
<onDetection>
 <action expressionLanguage="java">
  Queue myQueue = (Queue)act_lib.getExternalContext("OutputQueueOne");
  myQueue.enqueue(act_event);
 </action>
</onDetection>
```

Initializing and accessing variables

This example indicates how you can initialize and access rule set, rule block, or rule variables.

A variable can be defined at the level of the rule set, a rule block, or a rule. Before a variable can be accessed, it must be initialized with an initialization expression. The following expression initializes two variables, one named hostsList and one named hostsString:

```
<variable name="hostsList" dataType="java.util.ArrayList">
  <varInitializer expressionLanguage="java">
    return new ArrayList();
  </varInitializer>
</variable>
<variable name="hostsString" dataType="java.lang.String">
  <varInitializer expressionLanguage="java">
    return new String();
  </varInitializer>
</variable>
```

All variables are accessed through expressions. The following example shows how the variables hostsList and hostsString, which were initialized in the preceding example, are accessed through an expression within a rule response action. In this example, hostsList is modified, and hostsString is given a new value.

```
<onNextEvent>
 <action expressionLanguage="java">
   String hostname = act_event.getStringAttribute("hostname");
    ArrayList hostsList = (ArrayList)act_lib.getVariable("hostsList");
    hostsList.add(hostname);
    String hostsString = act_lib.getStringVariable("hostsString");
```

```
    String newHostString = hostsString + ", " + hostname);
    act_lib.setStringVariable("hostsString", newHostsString);
 </action>
</onNextEvent>
```

Accessing event-related information

The following examples indicate how you can access event-related information through the variables that are provided by the Active Correlation Technology.

Example of accessing the current event:
The following code shows how to use the act_event variable to obtain the hostname attribute for an event:

```
act_event.getAttribute("hostname");
```

Example of accessing events through the event list by index:
The following code shows how to use the act_eventList variable to obtain the first event in the event list:

```
act_eventList.get(0);
```

Example of accessing events through the event list by alias:
Unlike other rule types, the sequence rule allows multiple event selectors and actually requires a minimum of two event selectors. The alias attribute on the <eventSelector> element is valid only within a sequence rule, and it uniquely names an event that is selected by a certain event selector in the sequence rule. In an expression within a filtering predicate or action, you can use the act_eventList variable to access an event in a sequence rule by its alias name.

The following code shows two event selectors for a sequence rule. The alias names are TECevent and WASevent.

```
<eventSelector alias="TECevent">
 <eventType type="serverStatus"/>
 <filteringPredicate expressionLanguage="java">
  return act_event.getStringAttribute("source").equals("TEC");
 </filteringPredicate>
</eventSelector>
<eventSelector alias="WASevent">
 <eventType type="serverStatus"/>
 <filteringPredicate expressionLanguage="java">
  return act_event.getStringAttribute("source").equals("WAS");
 </filteringPredicate>
</eventSelector>
```

The following code shows how to use the act_eventList variable to obtain the event that has been accepted by the first event selector, which is named TECevent:

```
act_eventList.get("TECevent");
```

Best practices for coding expressions

This information includes some best practices, hints, and tips for coding expressions efficiently and effectively.

- For ease in understanding, most of the expression examples that are provided in this information include Java code directly within the XML constructs. However, when you create rules, the best practice is to use external modules to contain the Java code and to call these external modules as part of the expressions.

  You can also use or edit existing snippets, or create new snippets, in the rule builder to provide the code for the calls to external modules. Snippets are

excerpts of source code that can be used within expressions. In the rule builder, snippets are available through the Snippet view.

With this approach, the tasks of designing, developing, editing, testing, and debugging Java code can be done in the integrated development environment (IDE) of your choice and can be controlled as a regular part of your development process.

- To prevent expression code from being parsed as XML, enclose the code in a CDATA section, where `<![CDATA[` immediately precedes the code and `]]>` immediately follows the code, as shown in the following example:

```
<onTimeOut>
<action expressionLanguage="java">
<![CDATA[
  IEvent firstEvent = act_eventList.get(0);
  System.out.println("Expired Item: " + firstEvent.getAttribute("sourceComponentId.location"));
]]>
</action>
</onTimeOut>
```

  XML parsers ignore everything that is inside of a CDATA section.
- If you use the act_lib.getExternalContext() method in an expression, do not store the object that is returned from the method in a rule set or rule block variable. The reason is that an application can change the reference to the object instance, and the associated rule set or rule block variable is not updated.
- If you use a return statement (`return;`) in an expression within an <action> element, and additional <action> elements are coded for the respective rule response or life cycle action, the code execution ends where the return statement is coded and begins again in the expression within the next <action> element.
- Rule management and other Active Correlation Technology engine methods cannot be called from within a rule response or life cycle action.

Related information

"Importing and accessing external modules and objects" on page 20
This example indicates how you can make external code (such as Java classes) and external objects accessible to expressions. An external object is an object that an application creates to communicate with expressions.

"Including snippets in expressions within rules" on page 40
In expressions within rules, you can include snippets from the Snippet view in the Eclipse Workbench.

Variables

In the rule language, certain variables are used to store event-related information across different event occurrences or rules. This event-related information can then be accessed from expressions within rules. Some types of variables are defined by the rule writer, and others are provided by the Active Correlation Technology. Some types can be accessed directly within an expression, and others can be accessed only through methods that are provided by the Active Correlation Technology.

Variables defined within <variable> elements and accessed through methods

You can define a variable within the <variable> element for a rule, rule block, or rule set. You can then access this variable within an expression by using one of the following methods:

- The getVariable() method or one of the get*javatype*Variable() methods
- The setVariable() method or one of the set*javatype*Variable() methods

For example, if you define the variable *rule_writer_variable* within the <variable> element for a rule, you can access this variable with the following code:

```
int sample_variable = act_lib.getIntVariable("rule_writer_variable");
```

Variables provided by the Active Correlation Technology and accessed directly within an expression

The following variables are provided by the Active Correlation Technology. You can use these variables inline within an expression.

- act_event
- act_eventList
- act_lib

For example, with the following code, you can access the act_event variable to obtain the hostname attribute of an event:

```
act_event.getAttribute("hostname");
```

Variables provided by the Active Correlation Technology and accessed through methods

The following variables are provided by the Active Correlation Technology. You can access these variables within an expression by using the getVariable() method or one of the get*javatype*Variable() methods.

- act_eventCount
- act_location
- act_nodeName
- act_threshold

For example, with the following code, you can access the act_eventCount variable:

```
int eventcount_integer = act_lib.getIntVariable(IACTLibrary.EVENTCOUNT);
```

Table 3 shows the constants that the IACTLibrary interface provides for these variables. In your code, to ensure that any spelling or typographical errors are found during compilation rather than during run time, always use the constants that represent these variables rather than the variables themselves. For example, use `act_lib.getIntVariable(IACTLibrary.EVENTCOUNT);` rather than `act_lib.getIntVariable("act_eventCount");`.

Table 3. Variables with associated constants

Variable	Associated constant
act_eventCount	EVENTCOUNT
act_location	LOCATION
act_nodeName	NODENAME
act_threshold	THRESHOLD

Related reference

"variable element" on page 109
The <variable> element defines a variable and contains information in a form that can be referenced by expressions. A variable can be defined at the level of the rule set, a rule block, or a rule.

Data types for Active Correlation Technology variables

The variables that are provided by the Active Correlation Technology have different data types.

Table 4 shows the data types for these variables.

Table 4. Data types for Active Correlation Technology variables

Variable	Data type
act_ event	The type that is defined by the com.ibm.correlation.IEvent interface
act_ eventCount	int
act_ eventList	The type that is defined by the com.ibm.correlation.IEventList interface
act_ lib	The type that is defined by the com.ibm.correlation.IACTLibrary interface
act_ location	java.lang.String
act_ nodeName	java.lang.String
act_ threshold	Because this variable is the value of the threshold attribute on the <computedThreshold> or <eventCountThreshold> element of a threshold rule, the data type must be the same as the data type for the threshold attribute value.

Expression contexts in which variables are valid

The variables that are provided by the Active Correlation Technology are valid only in specific expression contexts.

Table 5 lists the expression contexts in which these variables are valid. The table shows an X in the column for each variable that is valid within the respective expression context. Additional usage restrictions that apply to these variables are described in the topic for the respective variable.

Table 5. Expression contexts in which variables are valid

Expression context	act_ event	act_ eventCount	act_ eventList	act_ lib	act_ location	act_ nodeName	act_ threshold
<action> within <onActivation>	X			X	X	X	X
<action> within <onDeactivation>	X	X	X	X	X	X	X
<action> within <onDetection>	X	X	X	X	X	X	X
<action> within <onLoad>				X	X	X	X
<action> within <onNextEvent>	X	X	X	X	X	X	
<action> within <onTimeOut>		X	X	X	X	X	X
<action> within <onTimeWindowComplete>		X	X	X	X	X	
<action> within <onUnload>				X	X	X	X
<booleanThreshold>	X	X	X	X	X	X	
<computedThreshold>	X	X	X	X	X	X	X
<computedValue>	X			X	X	X	
<computeFunction>	X	X	X	X	X	X	
<filteringPredicate>	X	X	X	X	X	X	X
<varInitializer> for a rule				X	X	X	X

act_event variable

The act_event variable provides access to methods that apply to the current event.

Details

Because a timer rule does not process events, the act_event variable within a timer rule applies only to the events that activate or deactivate the rule.

The act_event variable applies within the <onActivation> and <onDeactivation> actions only if an event activated or deactivated the rule. Otherwise, this variable is null.

Coding example

The following code accesses the act_event variable to obtain the hostname attribute of an event:

```
String host = act_event.getStringAttribute("hostname");
```

Methods that can be accessed

The methods to which the act_event variable provides access are defined in the IEvent interface, as shown in Table 6.

Table 6. IEvent interface with corresponding methods and location of Javadoc method descriptions

Interface	Methods	Location of Javadoc method descriptions
IEvent	• get • getAttribute • getBooleanAttribute • getByteAttribute • getShortAttribute • getIntAttribute • getLongAttribute • getFloatAttribute • getDoubleAttribute • getStringAttribute • set • getTimeStamp • setTimeStamp • getType • getOriginal	com.ibm.correlation.IEvent

act_eventCount variable

The act_eventCount variable is an integer equal to the number of events that have been accepted by a rule.

Details

For a duplicate rule, the value of the act_eventCount variable is the total number of accepted events, which includes both the original event and any duplicates. For

all other rule types, the value is the same as the size of the event list, which can be obtained through the act_eventList variable using the act_eventList.size() method.

The act_eventCount and act_eventList variables are not valid within a timer rule because a timer rule does not process events.

If a rule is defined with a grouping key, the act_eventCount, act_eventList, and act_threshold variables are not valid within the following expression contexts:

- Life cycle actions
- <filteringPredicate> within <activateOnEvent> or <deactivateOnEvent> within <activationInterval>
- <computedValue>

This is because in this case, the rule variables apply only to a rule instance, and rule instances do not exist at the time that these expressions are run.

Coding example

The following code accesses the act_lib variable to obtain the number of events that have been accepted by a rule:

```
int eventCt = act_lib.getIntVariable(IACTLibrary.EVENTCOUNT);
```

act_eventList variable

The act_eventList variable provides access to methods that apply to the list of events that have been accepted by a rule.

Details

Both a filter rule and a duplicate rule always have a list of no more than one event because a filter rule is a stateless rule and because a duplicate rule retains only the first analyzed event.

The act_eventCount and act_eventList variables are not valid within a timer rule because a timer rule does not process events.

If a rule is defined with a grouping key, the act_eventCount, act_eventList, and act_threshold variables are not valid within the following expression contexts:

- Life cycle actions
- <filteringPredicate> within <activateOnEvent> or <deactivateOnEvent> within <activationInterval>
- <computedValue>

This is because in this case, the rule variables apply only to a rule instance, and rule instances do not exist at the time that these expressions are run.

Coding example

The following code accesses the act_eventList variable to obtain the second event in the event list:

```
IEvent second_event = act_eventList.get(1);
```

Methods that can be accessed

The methods to which the act_eventList variable provides access are defined in the IEventList interface, as shown in Table 7 on page 28.

Table 7. IEventList interface with corresponding methods and location of Javadoc method descriptions

Interface	Methods	Location of Javadoc method descriptions
IEventList	• get • size • isEmpty • listIterator	com.ibm.correlation.IEventList

act_lib variable

The act_lib variable provides access to library methods in the Active Correlation Technology.

Details

The methods that the act_lib variable can access vary depending on the rule language element that contains the expression where the variable is used. See Table 8.

Table 8. Methods that the act_lib variable can access based on the context of its containing expression

Expression context	IACTLibrary methods	IExitableActionLibrary methods	IActionLibrary methods
<action> within <onActivation>	X		
<action> within <onDeactivation>	X		
<action> within <onDetection>	X	X	X
<action> within <onLoad>	X		
<action> within <onNextEvent>	X	X	X
<action> within <onTimeOut>	X		X
<action> within <onTimeWindowComplete>	X		X
<action> within <onUnload>	X		
<booleanThreshold>	X		
<computedThreshold>	X		
<computedValue>	X		
<computeFunction>	X		
<filteringPredicate>	X		
<varInitializer>	X		

Coding example

The following code accesses the act_lib variable to call the method that exits the rule set:

```
act_lib.exitRuleSet();
```

Methods that can be accessed

The methods to which the act_lib variable provides access are defined in the IACTLibrary, IExitableActionLibrary, and IActionLibrary interfaces, as shown in Table 9 on page 29.

Table 9. Interfaces with corresponding methods and location of Javadoc method descriptions

Interface	Methods	Location of Javadoc method descriptions
IACTLibrary	• trace • getVariable • getBooleanVariable • getShortVariable • getIntVariable • getLongVariable • getFloatVariable • getDoubleVariable • getStringVariable • setVariable • setBooleanVariable • setShortVariable • setIntVariable • setLongVariable • setFloatVariable • setDoubleVariable • setStringVariable • getExternalContext	com.ibm.correlation.IACTLibrary
IActionLibrary	• forward • forwardOnCompletion • activate • deactivate The methods that are defined in the IACTLibrary interface are also available to the IActionLibrary interface.	com.ibm.correlation.IActionLibrary
IExitableActionLibrary	• exitRuleSet • exitRuleBlock The methods that are defined in the IACTLibrary and IActionLibrary interfaces are also available to the IExitableActionLibrary interface.	com.ibm.correlation.IExitableActionLibrary

act_location variable

The act_location variable is a string that identifies the location, within the rule hierarchy, of an expression.

Details

The location is a fully qualified name that indicates the position of the expression in the rule hierarchy. It has the form *identifier.identifier...*, where each occurrence of *identifier* is one of the following:

- The value of the name attribute for an XML element that is in the respective hierarchy.
- For elements that occur multiple times in a rule block or rule and have no name attribute: the XML element that contains the expression, followed by an index

number in brackets. This index number indicates the position of the expression within its containing element. The counter for assigning index numbers begins with 0 rather than 1. Therefore, if an element is contained within the third <action> element, for example, the index number is shown as `action[2]`.

These identifiers are in descending order from the highest level rule block to the lowest level element that contains the expression.

Coding example

The following code accesses the act_lib variable to obtain the location of the expression:

```
String location = act_lib.getStringVariable(IACTLibrary.LOCATION);
```

Examples of the location that is returned from the variable

The following values are examples of the location that is returned from the act_location variable.

`ruleBlockA.ruleA.eventSelector[3].filteringPredicate`

This expression is contained within the following:

- The rule block with a name attribute value of `ruleBlockA`
- The rule with a name attribute value of `ruleA`
- The fourth <eventSelector> element
- The <filteringPredicate> element

`ruleBlockA.ruleA.onDetection.action[5]`

This expression is contained within the following:

- The rule block with a name attribute value of `ruleBlockA`
- The rule with a name attribute value of `ruleA`
- The <onDetection> element
- The sixth <action> element

`ruleBlockA.ruleA.variableA.varInitializer`

This expression is contained within the following:

- The rule block with a name attribute value of `ruleBlockA`
- The rule with a name attribute value of `ruleA`
- The variable with a name attribute value of `variableA`
- The <varInitializer> element

act_nodeName variable

The act_nodeName variable is a string that identifies the fully qualified name of a node.

Details

In the Active Correlation Technology, a node is an object within the rule hierarchy that can be individually and independently added to, removed from, or replaced within a rule set. Specifically, these objects are nodes: rules, rule blocks, rule block variables, and rule set variables. Because an object cannot be operated on individually and independently below the rule level, a rule variable is not a node.

The fully qualified node name for a rule with a name attribute value of `rule1` that is located within a rule block with a name attribute value of `ruleBlockA` is

`ruleBlockA.rule1`. Because rules are organized hierarchically within a rule set, a period (.) is used within a fully qualified node name to denote a descent into a lower level node.

Coding example

The following code accesses the act_nodeName variable to obtain the fully qualified name of a node:

```
String nodeName = act_lib.getStringVariable(IACTLibrary.NODENAME);
```

act_threshold variable

The act_threshold variable is the value of the threshold attribute, which is the defined threshold value, on the <computedThreshold> or <eventCountThreshold> element of a threshold rule.

Details

The act_threshold variable is valid only within a threshold rule.

If a rule is defined with a grouping key, the act_eventCount, act_eventList, and act_threshold variables are not valid within the following expression contexts:

- Life cycle actions
- <filteringPredicate> within <activateOnEvent> or <deactivateOnEvent> within <activationInterval>
- <computedValue>

This is because in this case, the rule variables apply only to a rule instance, and rule instances do not exist at the time that these expressions are run.

Coding example

The following code accesses the act_lib variable to obtain a defined threshold value:

```
int threshold = act_lib.getIntVariable(IACTLibrary.THRESHOLD);
```

Event flow through a rule set

Events flow through a rule set in the order that the rule blocks and rules are coded. When the Active Correlation Technology engine receives an event, the engine determines the event type and identifies the rules that use this event type for rule activation, event processing, or rule deactivation.

How events are used by rules

Each rule that uses the event first determines whether the event meets all of the specified criteria for rule activation, event processing, or rule deactivation. If it does, the rule takes the following action:

For rule activation
The actions within the <onActivation> element for the rule are run, if they are coded.

For event processing
The rule processes the event. When the rule pattern is matched, the rule response actions are run, if they are coded. In some situations, rule response actions can do the following:

- The action can cause the event to skip the processing in the remainder of the rule block or rule set.
- The action can send a new or existing event to another rule or rule block for processing.

For rule deactivation
The actions within the <onDeactivation> element for the rule are run, if they are coded.

Methods that can influence the flow of events

The Active Correlation Technology provides the following methods that can be called to influence the flow of events through the rule set. These methods are available through the act_lib variable.

exitRuleSet
This method specifies that the current event is not processed by any additional rules in the rule set.

exitRuleBlock
This method specifies that the current event is not processed by any additional rules in the current rule block or in any rule blocks that this rule block contains. However, it is processed by additional rules that are outside the scope of the current rule block.

forward
This method specifies that an event is to be sent to other rules and rule blocks even though the current rule has not completed its processing. Each of the other rules and rule blocks then processes the event completely before returning the event to the rule that called the forward method.

forwardOnCompletion
This method specifies that an event is to be sent to other rules and rule blocks after the current rule has completed its processing.

Chapter 3. Rule writing overview

Before writing rules to correlate events, you must understand and plan for event correlation and design the rules. You can use the Active Correlation Technology rule builder to write the rules and compile the rule set.

The Active Correlation Technology rule builder is a GUI for writing rules. It includes online help. In the rule builder, the resulting rule set file is an XML file with a file type of actl.

The Active Correlation Technology does not provide a change management or version control system.

Planning for event correlation

Planning for event correlation includes understanding, or learning about, what event correlation is and how it can be applied in your application.

Ensure that you understand the following concepts:

- The information in Chapter 1, "Introduction," on page 3 and Chapter 2, "Rule language overview," on page 5
- The events that your application processes

Each application can process a different set of events, as described in the following examples:

Insurance business example
In an insurance business, events that track the flow of work through the claims process can be generated and correlated to determine whether the business processes are completing in a timely manner.

Sales example
In a different type of business, sales results can be periodically summarized, reported, and compared to a goal to indicate the status of achieving the sales goals for a certain time period.

IT environment example
In an IT environment, a mission-critical system can generate an event every minute to indicate that a database server is running normally. Correlation rules can be written to monitor the receipt of these heartbeat events and to take certain rule response actions if an expected heartbeat event is not received.

You should also understand the format of the events that your application processes. The Active Correlation Technology provides Java classes and methods to access data within the events that are being processed by the Active Correlation Technology engine. However, a basic understanding of the underlying event objects is important if you want to use these classes and methods to access or change the events as they are being processed.

To plan for event correlation, take the following steps:

1. Determine the events from your application that you want to correlate.
2. Determine the rule patterns for correlating the events.

A rule pattern represents a specific event correlation situation and can be used to correlate events that contribute in some way to that situation. Think about how the events that your application processes relate to the rule patterns that are defined by the Active Correlation Technology rule language. This can help you determine the rule patterns that you need to use.

Always use the most appropriate pattern for your event correlation situation. For example, if you want a rule to detect a certain sequence of events, do not write code to include the sequence pattern behavior in the rule response actions for a filter rule. Rather, use the sequence pattern to create a sequence rule.

3. Identify the constructs of each rule pattern that you want to use.

   The following information summarizes the primary constructs in the rule language, although the details of each of these are unique to the rule pattern. This information is organized in approximately the same way that it is presented through the rule builder GUI:

   Characteristics
   The definition of the rule characteristics, including the rule name, description, and pattern. For details, see the following topics:
   - "collectionRule element" on page 71
   - "computationRule element" on page 72
   - "duplicateRule element" on page 79
   - "filterRule element" on page 87
   - "sequenceRule element" on page 101
   - "thresholdRule element" on page 105
   - "timerRule element" on page 107

   Variables
   The definition of rule variables, including the name, type, description, and initialization expression for each variable. For details, see "variable element" on page 109.

   Event selection
   The definition of the criteria that determines which events are accepted for processing by the rule. For details, see "eventSelector element" on page 84.

   Grouping key
   The definition of the grouping key, which is the way to direct the rule to create a separate rule instance (or copy of itself) for each group of events that share common characteristics. For details, see "groupingKey element" on page 88.

   Pattern specifics
   The specification of the time period during which the stateful rule is processing to match its pattern and the definition of unique aspects of certain stateful rule patterns. For details, see "timeWindow element" on page 109.

   For the computation rule, this includes the definition of the computation to apply to the collected events. For details, see "computeFunction element" on page 76.

   For the threshold rule, this includes the definition of the threshold type and other information that is specific to the threshold type. For details, see the following topics:
   - "booleanThreshold element" on page 70

- "computedThreshold element" on page 74
- "eventCountThreshold element" on page 81

Rule responses
The definition of the actions to take when the rule completes its processing.

For details, see the following topics:
- For duplicate, filter, sequence, or threshold rules: "onDetection element" on page 93
- For duplicate rules: "onNextEvent element" on page 95
- For sequence or threshold rules: "onTimeOut element" on page 95
- For collection, computation, duplicate, or timer rules: "onTimeWindowComplete element" on page 96

Activation interval
The definition of when a rule is active and inactive. For details, see "activationInterval element" on page 65.

Life cycle
The definition of the actions, if any, to take at the following four primary stages in the life cycle of a rule: load, activation, deactivation, and unload. Typically, these actions do not need to be defined. For details, see "lifeCycleActions element" on page 91.

4. Identify Java methods and associated snippets that are to be called within rule expressions. Rather than writing extensive Java code within rule expressions, rule writers should use Java methods to call external modules. These external modules can be provided by the application that embeds the Active Correlation Technology or can be created by the rule writer as necessary. The snippets that are associated with each of the Java methods should also be identified. For more information, see "Best practices for coding expressions" on page 22.

Proceed to "Designing the rules to correlate events."

Designing the rules to correlate events

After you plan for event correlation, you must design the rules to correlate the events.

First, complete "Planning for event correlation" on page 33.

To design the rules, take the following steps:

1. Design the organization of the rules and rule blocks within the rule set.
2. Design the level at which different variables are to be defined, for example, at the level of the rule set, rule block, or rule.
3. Design the elements of each rule based on the rule pattern.

   This step makes use of the constructs of each rule pattern that you identified in the planning stage.
4. Design the interactions among rules, especially with regard to the forwarding of events and the skipping of rule set processing for duplicate events.

   For more details, see "Event flow through a rule set" on page 31.
5. Design, develop, and test any Java methods and associated snippets that you created to be called within expressions.

Proceed to "Getting started with the rule builder" on page 36.

Getting started with the rule builder

After you design the rules to correlate events, you can use the Active Correlation Technology rule builder to write the rules.

The following steps are the primary steps in writing rules with the rule builder.

1. Open the Eclipse Workbench.
2. Set your perspective in the Eclipse Workbench.
3. Set your preferences for the Active Correlation Technology, or accept the default preferences.
4. Either create a project in which to store the rule set file, or use an existing project.
5. Create a rule set file, and store it in the project of your choice.
6. Create at least one rule block within the rule set. Additionally, you can create other rule blocks within the rule set, and you can create rule blocks within rule blocks.
7. Create rules within rule blocks.
8. Validate the rule set.
9. Compile the rule set.
10. Update the rule set as necessary.

In expressions within rules, you can also include snippets from the Snippet view in the Eclipse Workbench.

Setting the perspective in the Eclipse Workbench

Before you do anything else, you should set your perspective in the Eclipse Workbench. This topic describes how to open the Rule Builder perspective.

First, open the Eclipse Workbench, if it is not already open.

Although you can use a perspective other than the Rule Builder perspective, the steps for performing various tasks vary slightly depending on the perspective you choose.

To open the Rule Builder perspective, take the following steps:

1. In the Workbench, click **Window** → **Open Perspective** → **Other...**.
2. Click **Rule Builder**, and click **OK**. The Rule Builder perspective is then shown.

Proceed to "Setting preferences."

Setting preferences

Before you create a rule set file, you should ensure that your preferences for the Active Correlation Technology are set correctly in the Eclipse Workbench. This topic describes how to set these preferences.

First, open the Eclipse Workbench, if it is not already open.

To set your preferences for the Active Correlation Technology, take the following steps:

1. In the Workbench, click **Window** → **Preferences...**.

2. Click **Active Correlation Technology**, and on the Active Correlation Technology page, specify whether you want to add "act" as the prefix for newly created rules and rule blocks in the rule builder. By default, "act" is *not* added as the prefix.
3. Expand **Active Correlation Technology**. Depending on your application, the following additional items might be shown. Click these items to set the associated preferences.

Item	Associated preference
Common Base Event Definition Provider	You can specify the XML files that provide the structure for one or more event types (including the names and types of attributes that a given event type can contain) that conform to the Common Base Event specification. These event types and attributes are then available when you create rules.
Compiler	You can specify the following primary items: • Whether compiled rule sets should be saved • The file type for compiled rule sets • The class path for code that is to be used at compile time By default, compiled rule sets are saved and shown in the Navigator view with a file type of `acts`, which indicates that the compiler output is a serialized rule set.

Proceed to "Creating a project for storing a rule set file."

Creating a project for storing a rule set file

In the Eclipse Workbench, when you create a rule set file, you must specify the project in which to store the file. Therefore, you must either create a project before you create a rule set file, or use an existing project. This topic describes a way to create a project in the Eclipse Workbench.

First, open the Eclipse Workbench, if it is not already open. Also, open the Rule Builder perspective.

To create a simple project within the Workbench, take the following steps:

1. Click **File** → **New** → **Project...**.
2. In the New Project wizard, click **Simple** → **Project**, and click **Next**.
3. In the **Project name** field, type a unique name for the project. Also, either use the default location for the project, or choose a different location.
4. Click **Finish**. The new project is then shown in the Navigator view of the Rule Builder perspective.

Proceed to "Creating a rule set."

Creating a rule set

This topic describes how to create a rule set.

First, open the Eclipse Workbench, if it is not already open. Also, open the Rule Builder perspective.

To create a rule set file, take the following steps:

1. Click **File** → **New** → **Rule Set File**.
2. In the New Rule Set File wizard, click the name of the folder that is associated with the project in which you want to store the rule set file. This is probably the project that you created in "Creating a project for storing a rule set file" on page 37. The folder name is then shown in the first field.
3. In the **File name** field, type a name for the rule set file. The file type must be actl.
4. Click **Next**.
5. Type a unique name for the rule set and an optional description. If you do not want to use the default value for the **XML encoding** field, also specify the encoding style for the rule set file, which is an XML file.
6. Click **Finish**. The rule set file is then shown in its project folder in the Navigator view. Because the file is validated when it is saved, the word "Validated" is shown beside the file. The rule set file is also shown in the Outline view.

Proceed to "Creating a rule block."

Creating a rule block

This topic describes how to create a rule block within a rule set or within another rule block.

First, open the Eclipse Workbench, if it is not already open. Also, open the Rule Builder perspective.

If you are creating a rule set for the first time, you must create at least one rule block within the rule set before you can create any rules. After you have created that first rule block, you can create additional rule blocks, including rule blocks within rule blocks.

To create a rule block, take the following steps:

1. In the Navigator view, double-click the name of the rule set file that you want to update. The file is then opened in the Outline view. When you click the rule set in the Outline view, the current information for the rule set is shown in the editor area.
2. In the Outline view, right-click the rule set.
3. Click **New Child** → **Rule Block**. A rule block is then shown within the rule set in the Outline view, and the current information for the rule block is shown in the editor area.

You can now create additional rule blocks in the following ways:

- To create a rule block that is a peer to an existing rule block, right-click the existing rule block, and click **New Sibling** → **Rule Block**.
- To create a rule block within an existing rule block, right-click the existing rule block, and click **New Child** → **Rule Block**.

Also, using the editor, you can update the information for the rule set and each rule block. Proceed to "Creating a rule" on page 39.

Creating a rule

This topic describes how to create a rule.

First, open the Eclipse Workbench, if it is not already open. Also, open the Rule Builder perspective.

A rule must be created within a rule block. To create a rule, take the following steps:

1. In the Outline view, right-click the rule block that you want to update.
2. Click **New Child** and the type of rule that you want to create. The rule is then shown within the rule block in the Outline view, and the current information for the rule is shown in the editor area.

You can add additional rules to rule blocks in the same way. Also, using the editor, you can update the information for each rule. Proceed to "Validating a rule set."

Validating a rule set

This topic describes how to validate a rule set before you compile it.

First, open the Eclipse Workbench, if it is not already open. Also, open the Rule Builder perspective.

To validate a rule set, take the following steps:

1. In the Outline view, right-click a rule, a rule block, or the rule set.
2. Click **Validate Rule Set**. When the validation is complete, a message window is shown to indicate whether problems exist. If the validation completes successfully, the word "Validated" is shown to the right of the file name for the rule set in the Navigator view.

When the validation completes successfully, proceed to "Compiling a rule set."

Compiling a rule set

This topic describes how to compile a rule set.

First, open the Eclipse Workbench, if it is not already open. Also, open the Rule Builder perspective.

In either the Navigator or Outline view, right-click the rule set that you want to compile, and click **Compile Rule Set**. Any compilation errors are shown in the Problems view.

By default, if no compilation errors exist, the compiled rule set is shown in the Navigator view with a default file type of `acts`, which indicates that it is a serialized rule set. Also in the Navigator view, the word "Compiled" is shown to the right of the file name for the rule set.

Updating a rule set

This topic describes how to update a rule set.

First, open the Eclipse Workbench, if it is not already open. Also, open the Rule Builder perspective.

In the Outline view, click the item (rule, rule block, or rule set) that you want to update. The current information for the item is then shown in the editor area, and you can use the editor to update this information.

To create a rule block or rule that is a peer to an existing rule block or rule, take the following steps:

1. Right-click the existing rule block or rule.
2. Click **New Sibling** and the item (rule block or type of rule) that you want to add.

To create a rule block or rule within an existing rule block, take the following steps:

1. Right-click the existing rule block.
2. Click **New Child** and the item (rule block or type of rule) that you want to add.

To access other update functions in the Outline view, take the following steps:

1. Right-click the item (rule block or rule) that you want to update.
2. Click the menu item for the function, such as **Delete**, **Copy**, or **Paste**.

Including snippets in expressions within rules

In expressions within rules, you can include snippets from the Snippet view in the Eclipse Workbench.

First, open the Eclipse Workbench, if it is not already open. Also, open the Rule Builder perspective.

The Snippet view is shown in the Rule Builder perspective. Snippets are organized into categories based on their function.

To include a snippet from the Snippet view, take the following steps:

1. Click a snippet category to see the names of the snippets in the category.
2. Click the snippet that you want to include in an expression.
3. Drag the snippet into the respective **Expression** field. The code is placed at the position of the cursor in the **Expression** field. If the code requires input from the rule writer, such as specific message text or variable values, you are prompted to provide that input before the code is included in the expression.

Proceed to “Validating a rule set” on page 39.

Part 2. Rule Writer's Reference

Chapter 4. Summary of rule set organization

This reference lists all the language elements of a rule set, a rule block, and each type of rule. It serves as a quick reference for coding a rule set.

Table 10 explains the meaning of the notation that follows each language element. *n* represents an unbounded number.

Table 10. Explanation of notation that defines the number of occurrences for a language element

Notation	Meaning
(0, 1)	0 indicates that the language element is optional. 1 indicates that if coded, only 1 occurrence is allowed.
(0, *n*)	0 indicates that the language element is optional. *n* indicates that if coded, multiple occurrences are allowed.
(1, 1)	The first 1 indicates that the language element is required. The second 1 indicates that only 1 occurrence is allowed.
(1, *n*)	1 indicates that the language element is required. *n* indicates that multiple occurrences are allowed.
(2, *n*)	2 indicates that 2 occurrences of the language element are required. *n* indicates that additional occurrences are allowed.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Rule set summary

This summary lists the language elements of a rule set.

Rule set elements

<ruleSet> contains the following elements:

- <comment> (0, 1)
- <import> (0, n)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <ruleBlock> (0, n)

Related reference

"Rule block summary"
This summary lists the language elements of a rule block.

Rule block summary

This summary lists the language elements of a rule block.

Rule block elements

<ruleBlock> contains the following elements.

If they are coded, the <comment>, <import>, and <variable> elements must be coded in the order that is shown. The remaining elements can be coded in any order.

- <comment> (0, 1)
- <import> (0, n)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <ruleBlock> (0, n)
- <collectionRule> (0, n)
- <computationRule> (0, n)
- <duplicateRule> (0, n)
- <filterRule> (0, n)
- <sequenceRule> (0, n)
- <thresholdRule> (0, n)
- <timerRule> (0, n)

Related reference

"Collection rule summary"
This summary lists all language elements of the collection rule.

"Computation rule summary" on page 46
This summary lists all language elements of the computation rule.

"Duplicate rule summary" on page 47
This summary lists all language elements of the duplicate rule.

"Filter rule summary" on page 48
This summary lists all language elements of the filter rule.

"Sequence rule summary" on page 49
This summary lists all language elements of the sequence rule.

"Threshold rule summary" on page 51
This summary lists all language elements of the threshold rule.

"Timer rule summary" on page 52
This summary lists all language elements of the timer rule.

Collection rule summary

This summary lists all language elements of the collection rule.

Rule elements

<collectionRule> contains the following elements:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)

- <start> (0, 1)
  - One of the following three elements (1, 1):
    - <dateTime>
    - <whenLoaded>
    - <inactiveWhenLoaded>
- <stop> (0, 1)
  - One of the following three elements (1, 1):
    - <dateTime>
    - <never>
    - <after>

– <activateOnEvent> (0, 1)
- <eventType> (0, n)
- <filteringPredicate> (0, 1)

– <deactivateOnEvent> (0, 1)
- <eventType> (0, n)
- <filteringPredicate> (0, 1)

– <activationByGroupingKey> (0, 1)
- <activateOnEvent> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
  - <stopAfter> (0, 1)
- <deactivateOnEvent> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)

• <lifeCycleActions> (0, 1)
– <onLoad> (0, 1)
- <action> (0, n)

– <onActivation> (0, 1)
- <action> (0, n)

– <onDeactivation> (0, 1)
- <action> (0, n)

– <onUnload> (0, 1)
- <action> (0, n)

• <eventSelector> (0, 1)
– <eventType> (0, n)
– <filteringPredicate> (0, 1)

• <groupingKey> (0, 1)
– The following three elements, in any order (1, n):
- <attributeAlias>
  - <eventAttribute> (2, n)
- <attributeName>
- <computedValue>

• <timeWindow> (1, 1)
– One of the following two elements (1, 1):
- <timeInterval>

- <runUntilDeactivated>

- <onTimeWindowComplete> (0, 1)
  - <action> (0, n)

Computation rule summary

This summary lists all language elements of the computation rule.

Rule elements

<computationRule> contains the following elements:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <activationByGroupingKey> (0, 1)
    - <activateOnEvent> (0, 1)
      - <eventType> (0, n)
      - <filteringPredicate> (0, 1)
      - <stopAfter> (0, 1)
    - <deactivateOnEvent> (0, 1)
      - <eventType> (0, n)
      - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)

  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <groupingKey> (0, 1)
  - The following three elements, in any order (1, n):
    - <attributeAlias>
      - <eventAttribute> (2, n)
    - <attributeName>
    - <computedValue>
- <computeFunction> (1, 1)
- <timeWindow> (1, 1)
  - One of the following two elements (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onTimeWindowComplete> (0, 1)
  - <action> (0, n)

Duplicate rule summary

This summary lists all language elements of the duplicate rule.

Rule elements

<duplicateRule> contains the following elements:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)

        - <filteringPredicate> (0, 1)
    - <deactivateOnEvent> (0, 1)
        - <eventType> (0, n)
        - <filteringPredicate> (0, 1)
    - <activationByGroupingKey> (0, 1)
        - <activateOnEvent> (0, 1)
            - <eventType> (0, n)
            - <filteringPredicate> (0, 1)
            - <stopAfter> (0, 1)
        - <deactivateOnEvent> (0, 1)
            - <eventType> (0, n)
            - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
    - <onLoad> (0, 1)
        - <action> (0, n)
    - <onActivation> (0, 1)
        - <action> (0, n)
    - <onDeactivation> (0, 1)
        - <action> (0, n)
    - <onUnload> (0, 1)
        - <action> (0, n)
- <eventSelector> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
- <groupingKey> (0, 1)
    - The following three elements, in any order (1, n):
        - <attributeAlias>
            - <eventAttribute> (2, n)
        - <attributeName>
        - <computedValue>
- <timeWindow> (1, 1)
    - One of the following two elements (1, 1):
        - <timeInterval>
        - <runUntilDeactivated>
- <onDetection> (0, 1)
    - <action> (0, n)
- <onNextEvent> (0, 1)
    - <action> (0, n)
- <onTimeWindowComplete> (0, 1)
    - <action> (0, n)

Filter rule summary

This summary lists all language elements of the filter rule.

Rule elements

<filterRule> contains the following elements:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <onDetection> (0, 1)
  - <action> (0, n)

Sequence rule summary

This summary lists all language elements of the sequence rule.

Rule elements

<sequenceRule> contains the following elements:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <activationByGroupingKey> (0, 1)
    - <activateOnEvent> (0, 1)
      - <eventType> (0, n)
      - <filteringPredicate> (0, 1)
      - <stopAfter> (0, 1)
    - <deactivateOnEvent> (0, 1)
      - <eventType> (0, n)
      - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (2, n)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)

- <groupingKey> (0, 1)
  - The following three elements, in any order (1, n):
    - <attributeAlias>
      - <eventAttribute> (2, n)
    - <attributeName>
    - <computedValue>
- <timeWindow> (1, 1)
  - One of the following two elements (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onDetection> (0, 1)
  - <action> (0, n)
- <onTimeOut> (0, 1)
  - <action> (0, n)

Threshold rule summary

This summary lists all language elements of the threshold rule.

Rule elements

<thresholdRule> contains the following elements:

- <comment> (0, 1)
- <variable> (0, n)
  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <activationByGroupingKey> (0, 1)
    - <activateOnEvent> (0, 1)

        - <eventType> (0, n)
        - <filteringPredicate> (0, 1)
        - <stopAfter> (0, 1)
      - <deactivateOnEvent> (0, 1)
        - <eventType> (0, n)
        - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <eventSelector> (0, 1)
  - <eventType> (0, n)
  - <filteringPredicate> (0, 1)
- <groupingKey> (0, 1)
  - The following three elements, in any order (1, n):
    - <attributeAlias>
      - <eventAttribute> (2, n)
    - <attributeName>
    - <computedValue>
- One of the following three elements (1, 1):
  - <booleanThreshold>
  - <computedThreshold>
  - <eventCountThreshold>
- <timeWindow> (1, 1)
  - One of the following two elements (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onDetection> (0, 1)
  - <action> (0, n)
- <onTimeOut> (0, 1)
  - <action> (0, n)

Timer rule summary

This summary lists all language elements of the timer rule.

Rule elements

<timerRule> contains the following elements:

- <comment> (0, 1)
- <variable> (0, n)

  - <comment> (0, 1)
  - <varInitializer> (1, 1)
- <activationInterval> (0, 1)
  - <activationTime> (0, 1)
    - <start> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <whenLoaded>
        - <inactiveWhenLoaded>
    - <stop> (0, 1)
      - One of the following three elements (1, 1):
        - <dateTime>
        - <never>
        - <after>
  - <activateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
  - <deactivateOnEvent> (0, 1)
    - <eventType> (0, n)
    - <filteringPredicate> (0, 1)
- <lifeCycleActions> (0, 1)
  - <onLoad> (0, 1)
    - <action> (0, n)
  - <onActivation> (0, 1)
    - <action> (0, n)
  - <onDeactivation> (0, 1)
    - <action> (0, n)
  - <onUnload> (0, 1)
    - <action> (0, n)
- <timeWindow> (1, 1)
  - One of the following two elements (1, 1):
    - <timeInterval>
    - <runUntilDeactivated>
- <onTimeWindowComplete> (0, 1)
  - <action> (0, n)

Chapter 5. Language element reference

This reference describes the details of the language elements in the XML schema for the Active Correlation Technology rule language. The language elements are listed in alphabetical order, and the attributes that are available for each element are described within the topic for that element.

In XML, and in other markup languages such as SGML and HTML, an element is a basic unit that consists of a start tag, end tag, associated attributes and their values, and any text that is contained between the start and end tags. An attribute is a name-value pair that is coded on an element to define a certain characteristic of the element. An attribute has a data type that identifies the kind of information that is provided in its value (for example, numerical, textual, or boolean information).

In XML, a namespace is a uniform resource identifier (URI) that provides a unique name to associate with the elements and type definitions in a schema. The URI indicates which XML schema contains the definition of an element. A namespace is specified with a prefix string followed by a colon. The Active Correlation Technology rule language schema is defined in three different files and uses the following three namespaces:

xsd: This namespace indicates that the language element is defined in the standard XML schema, which is described at http://www.w3.org.

br: This namespace indicates that the language element is defined in the Active Correlation Technology base rule set schema, which is in the ACTparser.jar file in the com/ibm/correlation/ruleparser/xml/RuleSetBase.xsd subdirectory. For example, `br:ruleSet` refers to the ruleSet element that is defined in the RuleSetBase.xsd file.

act: This namespace indicates that the language element is defined in the Active Correlation Technology language schema, which is in the ACTparser.jar file in the com/ibm/correlation/ruleparser/xml/ACTL.xsd subdirectory. For example, `act:ruleSet` refers to the ruleSet element that is defined in the ACTL.xsd file.

In the rule language schema, language elements are defined as either elements or complex types, for example:

```
<xsd:element name="symbol" minOccurs="1" maxOccurs="unbounded"></element>
<xsd:complexType name="symbol"></complexType>
```

In the schema, the minOccurs and maxOccurs attributes define the minimum and maximum number of occurrences, respectively, for a language element. Table 11 describes the meaning of different values for the minOccurs and maxOccurs attributes.

Table 11. Attributes in the schema that define the number of occurrences for a language element

Attribute	Attribute value	Meaning
minOccurs	0	The language element is optional.
minOccurs	1	The language element must occur at least once. 1 is the default value for the minOccurs attribute.

Table 11. Attributes in the schema that define the number of occurrences for a language element (continued)

Attribute	Attribute value	Meaning
minOccurs	2	The language element must occur at least twice.
maxOccurs	1	The language element cannot occur more than once. 1 is the default value for the maxOccurs attribute.
maxOccurs	unbounded	The language element can occur any number of times.

action element

The <action> element contains an expression that defines either a rule response action or a life cycle action.

Details

Refer to "Variables" on page 23 for information about the variables that can be used in expressions. The use of certain variables is dependent on the context of the expression.

Attributes

<action> has the following attributes:

Table 12. Attributes of the <action> element

Name	Description	Data type	Required?
expressionLanguage	Identifies the programming language in which the expression is written. Because the Java programming language is the only supported expression language, the only valid value for this attribute is `java`.	xsd:NMTOKEN	Yes
name	Identifies the action. This identifier can be useful for troubleshooting purposes, especially if it is a unique name among all <action> elements that are defined for a specific rule response or life cycle action.	xsd:NMTOKEN	No

Contained within

<action> is contained within the following elements:

- <onActivation>
- <onDeactivation>
- <onDetection>
- <onLoad>
- <onNextEvent>
- <onTimeOut>
- <onTimeWindowComplete>
- <onUnload>

Contains

<action> contains no elements.

Related concepts

"Expressions" on page 19
An expression is code that contains custom logic that can be added to a rule. Expressions can also access code that is external to the Active Correlation Technology engine. In the rule language, expressions are valid only within specific contexts, or rule language elements.

activateOnEvent element

The <activateOnEvent> element defines the events that can activate the rule or, for rules that are defined with a <groupingKey> element, a rule instance.

Following are the three possible ways of selecting events:

- The use of one or more <eventType> elements with a <filteringPredicate> element
- The use of one or more <eventType> elements without a <filteringPredicate> element
- The use of a <filteringPredicate> element without any <eventType> elements

If the rule is inactive and no <eventType> or <filteringPredicate> element is coded, any event that occurs is selected.

Not coding any <eventType> elements can negatively impact system performance.

Assume that you want to select all events of type `Audit Failure`. You can use a filtering predicate to further refine the selection criteria to include only the events that have an event attribute with a certain value. For example, you would code an <eventType> element to select all events of type `Audit Failure`, and code a <filteringPredicate> element to select only those events that have a hostname attribute with the value `MyCriticalSystem`.

Attributes

<activateOnEvent> has no attributes.

Contained within

<activateOnEvent> is contained within the following elements:

- <activationInterval>
- <activationByGroupingKey>

Contains

<activateOnEvent> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 13. Elements contained within the <activateOnEvent> element

Element	Required or optional?
<eventType>	Optional. 0 or more occurrences are allowed.
<filteringPredicate>	Optional. 0 or 1 occurrence is allowed.
<stopAfter>	This element is valid only when the <activateOnEvent> element is contained within the <activationByGroupingKey> element. Optional. 0 or 1 occurrence is allowed.

activationByGroupingKey element

The <activationByGroupingKey> element contains elements that specify the events that can activate and deactivate a rule instance that is defined by the <groupingKey> element. Because the <groupingKey> element is not valid for the filter and timer rules, the <activationByGroupingKey> element does not apply to these rules.

Details

The function that is provided by the <activationByGroupingKey> element is for use in rules where you define a grouping key. It allows you to control the activation and deactivation of rule instances based on the grouping key. When you code the <activationByGroupingKey> element, each rule instance can be individually activated and deactivated based on the <activateOnEvent> and <deactivateOnEvent> conditions within <activationByGroupingKey>.

The following example illustrates the use of the <activationByGroupingKey> element within a computation rule.

- The following computation rule accepts events of type `StockSharesTraded`. These events indicate the number of stock shares that are being traded for many different companies.
- The grouping key is the stockSymbol attribute of an event. The value of the stockSymbol attribute is the name of a specific company.
- The value of the sharesTraded attribute of an event is the number of stock shares that have been traded for the respective company (the company is indicated by the value of the stockSymbol attribute).
- For a specific company, a report is created that indicates the number of stock shares that have been traded for that company during a 10-minute period. However, before the computation rule can create this report, it must receive an event of type `StartReporting`, with the name of the respective company as the value of the stockSymbol attribute.

```
<computationRule name="StockReporter">
 <variable dataType="java.lang.Integer" name="totalSharesTraded">
  <varInitializer expressionLanguage="java">
   return new Integer(0);
  </varInitializer>
 </variable>

 <activationInterval>
  <activationTime>
   <start>
    <inactiveWhenLoaded/>
   </start>
  </activationTime>
```

```
 <activationByGroupingKey>
  <activateOnEvent>
   <eventType type="StartReporting"/>
  </activateOnEvent>
 </activationByGroupingKey>
</activationInterval>

<eventSelector>
 <eventType type="StockSharesTraded"/>
</eventSelector>

<groupingKey>
 <attributeName>stockSymbol</attributeName>
</groupingKey>

<computeFunction assignTo="totalSharesTraded" expressionLanguage="java">
 return new Integer(act_lib.getIntVariable("totalSharesTraded")
           + act_event.getIntAttribute("sharesTraded"));
</computeFunction>

<timeWindow>
 <timeInterval unit="ISO-8601" duration="PT10M"/>
</timeWindow>

<onTimeWindowComplete>
 <action expressionLanguage="java">
  StockReport.createReport(act_eventList.get(0).getStringAttribute("stockSymbol"),
              act_lib.getIntVariable("totalSharesTraded"));
 </action>
</onTimeWindowComplete>
</computationRule>
```

Attributes

<activationByGroupingKey> has no attributes.

Contained within

<activationByGroupingKey> is contained within the following element:

- <activationInterval>

Contains

<activationByGroupingKey> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 14. Elements contained within the <activationByGroupingKey> element

Element	Required or optional?
<activateOnEvent>	Optional. 0 or 1 occurrence is allowed.
<deactivateOnEvent>	Optional. 0 or 1 occurrence is allowed.

Relationship between <activationInterval> and <activationByGroupingKey> elements

The <activateOnEvent> and <deactivateOnEvent> elements are contained within both of these elements:

- <activationInterval>
- <activationByGroupingKey>, which is also contained within <activationInterval>

Rule behavior can differ based on the current rule activity and on the interactions between the <activateOnEvent> and <deactivateOnEvent> definitions within the <activationInterval> and <activationByGroupingKey> elements. The following example illustrates how these definitions can interact.

In this example, a duplicate rule is defined to suppress events from systems that are in maintenance mode and to provide, at the end of the maintenance period, a summary report of the number of events that were suppressed.

By default, a rule within which a grouping key is defined allows all grouping key values to be processed. Therefore, when events meet the event selection criteria for the rule, all rule instances are active and ready to accept these events according to any value of the grouping key. The activation interval for the rule is the same as it would be if the rule did not have a grouping key because in essence, all events that meet the event selection criteria for the rule are processed.

In the following example, the grouping key is `hostname`, and the definitions within the <activationInterval> element specify the following actions:

1. Activate all rule instances when an event of type `StartMaintenanceModeAllHosts` is received.
2. Deactivate all rule instances either after 2 hours or when an event of type `StopMaintenanceModeAllHosts` is received.

```
<duplicateRule name="Maintenance_Supression">
  <activationInterval>
    <activationTime>
      <start>
        <inactiveWhenLoaded/>
      </start>
      <stop>
        <after duration="PT2H" unit="ISO-8601"/>
      </stop>
    </activationTime>
    <activateOnEvent>
      <eventType type="StartMaintenanceModeAllHosts"/>
    </activateOnEvent>
    <deactivateOnEvent>
      <eventType type="StopMaintenanceModeAllHosts"/>
    </deactivateOnEvent>
  </activationInterval>
  <groupingKey missingAttributeHandling="ignoreEvent">
    <attributeName>hostname</attributeName>
  </groupingKey>
  <timeWindow>
    <runUntilDeactivated/>
  </timeWindow>
  <onDetection>
    <action expressionLanguage="java" name="DropEvent">
      <![CDATA[act_lib.exitRuleSet();]]>
    </action>
  </onDetection>
  <onTimeWindowComplete>
    <action expressionLanguage="java" name="CreateSummaryOfSupressedEvents">
      <![CDATA[Helper.createSummaryEvent("MaintenanceSummary", act_eventList, act_lib);]]>
    </action>
  </onTimeWindowComplete>
</duplicateRule>
```

In some situations, you might want to control which rule instances become active and when they become active. For these situations, you should code the <activationByGroupingKey> element.

The following example extends the previous example and illustrates how you can use the grouping key value to select which rule instances are allowed to process. The definitions within the <activationByGroupingKey> element specify the following actions:

1. Allow the rule instances for specific host names to process when events of type `StartMaintenanceMode` are received for those host names.
2. Deactivate these rule instances either after 2 hours of activation or when an event of type `StopMaintenanceMode` is received for the respective host name.

```
<activationByGroupingKey>
  <activateOnEvent>
    <eventType type="StartMaintenanceMode"/>
    <stopAfter duration="PT2H" unit="ISO-8601"/>
  </activateOnEvent>
  <deactivateOnEvent>
    <eventType type="StopMaintenanceMode"/>
  </deactivateOnEvent>
</activationByGroupingKey>
```

The following statements summarize what happens when you code the <activationByGroupingKey> element:

- When the <activateOnEvent> element is coded within the <activationByGroupingKey> element, only the events that share the grouping key value of the event that met the <activationByGroupingKey> <activateOnEvent> condition are allowed to process.
- When the <deactivateOnEvent> element is coded within the <activationByGroupingKey> element, the events that share the grouping key value of the event that met the <activationByGroupingKey> <deactivateOnEvent> condition are *not* allowed to process.

The effect of different activation and deactivation definitions on rule state

Table 15 on page 62 and Table 16 on page 64 show how the state of a rule is affected by different activation and deactivation definitions. In these tables, the following conventions are used:

- *A* is an activating event.
- In the notation "*A[x]*, " *x* represents the grouping key value. For example, *A[1]* is an activating event with a grouping key value of 1.
- *D* is a deactivating event.
- In the notation "*D[x]*, " *x* represents the grouping key value. For example, *D[1]* is an deactivating event with a grouping key value of 1.

Table 15. Rule state changes based on different activation definitions

Starting rule state	Rule state is potentially affected by	Ending rule state
Inactive	Time defined within <activationInterval> <activationTime> <start>	1. The rule is activated. 2. The <onActivation> actions run. 3. All grouping key values are allowed.
	activate() method	
	Event *A* , defined within <activationInterval> <activateOnEvent>	
	Event *A[1]* , defined within <activationByGroupingKey> <activateOnEvent> (*without* <stopAfter>)	1. The rule is activated. 2. The <onActivation> actions run. 3. Only the grouping key value of 1 is allowed. When the rule pattern is matched for this rule instance, the grouping key value of 1 is no longer allowed.
	Event *A[2]* , defined within <activateOnEvent> (*with* <stopAfter>)	1. The rule is activated. 2. The <onActivation> actions run. 3. Only the grouping key value of 2 is allowed, and it is allowed only for the duration that is specified by the <stopAfter> element. The rule pattern for this rule instance can be matched multiple times during this duration.
• Active • Allowing all grouping key values	Time defined within <activationInterval> <activationTime> <start>	No change has occurred in the rule state. It is the same as the starting rule state.
	activate() method	
	Event *A* , defined within <activationInterval> <activateOnEvent>	
	Event *A[1]* , defined within <activationByGroupingKey> <activateOnEvent> (*without* <stopAfter>)	
	Event *A[2]* , defined within <activateOnEvent> (*with* <stopAfter>)	

Table 15. Rule state changes based on different activation definitions (continued)

Starting rule state	Rule state is potentially affected by	Ending rule state
• Active • Allowing only grouping key values that have triggered rule instances based on <activationByGroupingKey> <activateOnEvent> definitions	Time defined within <activationInterval> <activationTime> <start>	No change has occurred in the rule state. It is the same as the starting rule state.
	activate() method	
	Event *A* , defined within <activationInterval> <activateOnEvent>	All grouping key values are allowed.
	Event *A[1]* , defined within <activationByGroupingKey> <activateOnEvent> (*without* <stopAfter>)	• The grouping key value of 1 is now allowed, in addition to the grouping key values that were previously allowed. • When the rule pattern is matched for this rule instance, the grouping key value of 1 is no longer allowed.
	Event *A[2]* , defined within <activateOnEvent> (*with* <stopAfter>)	• The grouping key value of 2 is now allowed, in addition to the grouping key values that were previously allowed. • This value is allowed only for the duration that is specified by the <stopAfter> element. • The rule pattern for this rule instance can be matched multiple times during this duration.
• Active • Allowing all grouping key values except those that are not allowed based on <activationByGroupingKey> <deactivateOnEvent> definitions	Time defined within <activationInterval> <activationTime> <start>	No change has occurred in the rule state. It is the same as the starting rule state.
	activate() method	
	Event *A* , defined within <activationInterval> <activateOnEvent>	All grouping key values are allowed.
	Event *A[1]* , defined within <activationByGroupingKey> <activateOnEvent> (*without* <stopAfter>)	The grouping key value of 1 is now allowed, in addition to the grouping key values that were previously allowed.
	Event *A[2]* , defined within <activateOnEvent> (*with* <stopAfter>)	The grouping key value of 2 is now allowed, in addition to the grouping key values that were previously allowed.

Table 16. Rule state changes based on different deactivation definitions

Starting rule state	Rule state is potentially affected by	Ending rule state
Inactive	Time defined within <activationInterval> <activationTime> <stop>	No change has occurred in the rule state. It is the same as the starting rule state.
	deactivate() method	
	Event *D* , defined within <activationInterval> <deactivateOnEvent>	
	Event *D[1]* , defined within <activationByGroupingKey> <deactivateOnEvent>	
	Duration defined within <activationByGroupingKey> <activateOnEvent> <stopAfter> ends for event *A[2]*	
• Active • Allowing all grouping key values	Time defined within <activationInterval> <activationTime> <stop>	1. All rule instances are deactivated. 2. The <onDeactivation> actions run. 3. The rule is deactivated.
	deactivate() method	
	Event *D* , defined within <activationInterval> <deactivateOnEvent>	
	Event *D[1]* , defined within <activationByGroupingKey> <deactivateOnEvent>	• The grouping key value of 1 is no longer allowed. • If the rule instance with a grouping key value of 1 is active, it is deactivated.
	Duration defined within <activationByGroupingKey> <activateOnEvent> <stopAfter> ends for event *A[2]*	The rule instance with a grouping key value of 2 is deactivated.
• Active • Allowing only grouping key values that have triggered rule instances based on <activationByGroupingKey> <activateOnEvent> definitions	Time defined within <activationInterval> <activationTime> <stop>	1. All rule instances are deactivated. 2. The <onDeactivation> actions run. 3. The rule is deactivated.
	deactivate() method	
	Event *D* , defined within <activationInterval> <deactivateOnEvent>	
	Event *D[1]* , defined within <activationByGroupingKey> <deactivateOnEvent>	• The grouping key value of 1 is no longer allowed. • If the rule instance with a grouping key value of 1 is active, it is deactivated.
	Duration defined within <activationByGroupingKey> <activateOnEvent> <stopAfter> ends for event *A[2]*	• The grouping key value of 2 is no longer allowed. • The rule instance with a grouping key value of 2 is deactivated.

Table 16. Rule state changes based on different deactivation definitions (continued)

Starting rule state	Rule state is potentially affected by	Ending rule state
• Active • Allowing all grouping key values except those that are not allowed based on <activationByGroupingKey> <deactivateOnEvent> definitions	Time defined within <activationInterval> <activationTime> <stop>	1. All rule instances are deactivated. 2. The <onDeactivation> actions run. 3. The rule is deactivated.
	deactivate() method	
	Event *D* , defined within <activationInterval> <deactivateOnEvent>	
	Event *D[1]* , defined within <activationByGroupingKey> <deactivateOnEvent>	• The grouping key value of 1 is no longer allowed. • If the rule instance with a grouping key value of 1 is active, it is deactivated.
	Duration defined within <activationByGroupingKey> <activateOnEvent> <stopAfter> ends for event *A[2]*	The rule instance with a grouping key value of 2 is deactivated.

activationInterval element

The <activationInterval> element contains elements that define when a rule is active and inactive.

Details

A rule can be activated or deactivated at a discrete point in time or by a specific event.

If you specify that a rule is to be activated, or deactivated, at a discrete point in time *and* by a specific event, the rule is activated, or deactivated, by whichever occurs first, the point in time or the receipt of the event. However, in this case, the rule might be activated or deactivated by many events throughout its life cycle. For example, a rule might be activated by an event, deactivated, activated at a defined point in time, deactivated again, and activated by another event.

In a business environment, you might want to activate a rule when an event is received that indicates that the stock exchange has opened for business. In an IT environment, you might want to start a maintenance window at 06:00 on 29 October 2005 and end it at one of the following times, based on whichever occurs first:

- 11:30 on 30 October 2005
- When an event is received that indicates that the maintenance work is complete

Attributes

<activationInterval> has no attributes.

Contained within

<activationInterval> is contained within the following elements:

- <collectionRule>
- <computationRule>

- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>
- <timerRule>

Contains

<activationInterval> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 17. Elements contained within the <activationInterval> element

Element	Required or optional?
<activationTime>	Optional. 0 or 1 occurrence is allowed.
<activateOnEvent>	Optional. 0 or 1 occurrence is allowed.
<deactivateOnEvent>	Optional. 0 or 1 occurrence is allowed.
<activationByGroupingKey>	Optional. 0 or 1 occurrence is allowed.

Relationships among the contained elements

The <start> and <stop> elements that are contained within the <activationTime> element are a static method of activating and deactivating a rule. Through these elements, a rule is activated or deactivated at a discrete point in time. In contrast, the <activateOnEvent> and <deactivateOnEvent> elements are a dynamic method of activating and deactivating a rule. Through these elements, a rule is activated or deactivated if a certain event occurs. For example, a rule is activated by any event that meets the criteria that is defined for the <activateOnEvent> element, if the rule is not already active. A rule is deactivated by any event that meets the criteria that is defined for the <deactivateOnEvent> element, if the rule is not already inactive. Therefore, certain events can alter the static definition of when a rule is activated or deactivated.

Table 18 on page 67 describes how and when a rule is activated or deactivated based on certain combinations in which the following elements might be coded:

- <start>
- <stop>
- <activateOnEvent>
- <deactivateOnEvent>

In Table 18 on page 67, *X* represents the name of an event that activates the rule, and *Y* represents the name of an event that deactivates the rule.

If the <start> element is not coded at all, the default start time is the same as that defined by the <whenLoaded> element.

If the <stop> element is not coded at all, the default stop time is the same as that defined by the <never> element.

Table 18. Rule activity based on coding different combinations of the elements that are contained within <activationInterval>

<activationTime>		<activateOnEvent>	<deactivateOnEvent>	Rule activity
<start>	**<stop>**			
<whenLoaded>	<never>			Rule is active when it is loaded and remains active while the Active Correlation Technology engine is running.
<whenLoaded>	<never>		*Y*	Rule is active when it is loaded. Event *Y* deactivates the rule.
<whenLoaded>	<never>	*X*	*Y*	Rule is active when it is loaded. Event *Y* deactivates the rule, and event *X* reactivates it. This deactivation and reactivation can occur multiple times.
<whenLoaded>	<after>			Rule is active when it is loaded, and it is deactivated after a specified time interval.
<whenLoaded>	<dateTime>			Rule is active when it is loaded, and it is deactivated at a specified date and time.
<inactiveWhenLoaded>	<never>	*X*		Rule is inactive when it is loaded. Event *X* activates the rule, and the rule remains active while the Active Correlation Technology engine is running.
<inactiveWhenLoaded>	<never>	*X*	*Y*	Rule is inactive when it is loaded. Event *X* activates the rule, and event *Y* deactivates it. This activation and deactivation can occur multiple times.
<dateTime>	<dateTime>			Rule is activated at a specified date and time and deactivated at a specified date and time.
<dateTime>	<dateTime>	*X*	*Y*	Rule is activated at a specified date and time and deactivated at a specified date and time. Event *X* activates the rule, and event *Y* deactivates it. Event *X* and *Y* can activate and deactivate the rule multiple times.
<dateTime>	<never>			Rule is activated at a specified date and time and remains active while the Active Correlation Technology engine is running.
<dateTime>	<never>		*Y*	Rule is activated at a specified date and time. Event *Y* deactivates the rule.
<dateTime>	<never>	*X*	*Y*	Rule is activated at a specified date and time. Event *Y* deactivates the rule, and event *X* reactivates it. This deactivation and reactivation can occur multiple times.
<dateTime>	<after>			Rule is activated at a specified date and time and deactivated after a specified time interval.
<dateTime>	<after>	*X*	*Y*	Rule is activated at a specified date and time and deactivated after a specified time interval. Event *X* activates the rule, and event *Y* deactivates it. This activation and deactivation can occur multiple times.

activationTime element

The <activationTime> element defines discrete points in time when a rule is activated or deactivated.

Attributes

<activationTime> has no attributes.

Contained within

<activationTime> is contained within the following element:

- <activationInterval>

Contains

<activationTime> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 19. Elements contained within the <activationTime> element

Element	Required or optional?
<start>	Optional. 0 or 1 occurrence is allowed.
<stop>	Optional. 0 or 1 occurrence is allowed.

after element

The <after> element specifies the duration of time that the rule is to remain active once it becomes active. After this duration of time, the rule is to be deactivated.

Attributes

<after> has the following attributes:

Table 20. Attributes of the <after> element

Name	Description	Data type	Required?
duration	Specifies the amount of time for the duration. The data type of this attribute is dependent on the value of the unit attribute.	• If the value of the unit attribute is `ISO-8601`, the data type is xsd:duration. • If the value of the unit attribute is `milliseconds`, the data type is xsd:positiveInteger.	Yes
unit	Specifies the unit of time to use. The valid values for this attribute are: • `ISO-8601` • `milliseconds`	xsd:string	Yes

The use of the ISO 8601 standard for time duration

Coding `ISO-8601` as the value for the unit attribute indicates that the value of the duration attribute is coded according to the ISO 8601 standard for specifying a time duration as one string. The standard XML schema data type specification uses ISO 8601 to provide a data type called `duration`. This data type is described in detail at http://www.w3.org/TR/xmlschema-2/#duration.

The format for the `duration` data type in the standard XML schema is the following string:

`PnYnMnDTnHnMnS`

- `P` is the character that always begins the string.
- *n*`Y` represents the number of years. A year is the same as 365 days. Therefore, coding `1Y` is the same as coding `365D`.
- *n*`M` represents the number of months. A month is the same as 30 days. Therefore, coding `1M` is the same as coding `30D`.
- *n*`D` represents the number of days.

- `T` is a separator that separates day units (years, months, and days) from time units (hours, minutes, and seconds). Time units always follow `T`.
- *n*`H` represents the number of hours.
- *n*`M` represents the number of minutes.
- *n*`S` represents the number of seconds.

Following are examples of the format:

- `P5DT12H` is 5.5 days.
- `PT59M59S` is 59 minutes and 59 seconds.
- `P1M` is 1 month.

Contained within

<after> is contained within the following element:

- <stop>

Contains

<after> contains no elements.

attributeAlias element

The <attributeAlias> element provides an alias name to associate event attributes that have the same meaning but different names in different events. For example, three different events might use these three different names for an event attribute that indicates the name of the system that originates the event: host, hostname, and source. The <attributeAlias> element contains the <eventAttribute> elements that describe the individual event attributes that must be associated as one event attribute for the grouping key.

Details

The <attributeAlias> element and its aliasName attribute are valid only in the context of a grouping key. This element and its attribute cannot be referenced in any expression, including an expression within the <computedValue> element.

Attributes

<attributeAlias> has the following attribute:

Table 21. Attributes of the <attributeAlias> element

Name	Description	Data type	Required?
aliasName	Defines the name for the event attributes that are described in the <eventAttribute> elements and are to be associated as one event attribute for the grouping key. This name must be unique within the rule.	xsd:NMTOKEN	Yes

Contained within

<attributeAlias> is contained within the following element:

- <groupingKey>

Contains

<attributeAlias> contains the following element:

Table 22. Elements contained within the <attributeAlias> element

Element	Required or optional?
<eventAttribute>	2 occurrences of this element are required. Additional occurrences are allowed.

attributeName element

The <attributeName> element contains the name of a specific event attribute that is part of the grouping key. This name must match the name that is used in the getAttribute method call for the act_event variable.

Attributes

<attributeName> has no attributes.

Contained within

<attributeName> is contained within the following element:

- <groupingKey>

Contains

<attributeName> contains no elements.

booleanThreshold element

The <booleanThreshold> element is valid only for the threshold rule. It contains an expression that is called as each event is received. The expression computes or compares the threshold value based on the current event and on any other events that have been accepted by the rule. The expression returns a boolean value of `true` or `false` to indicate whether the threshold has been met.

Details

Refer to "Variables" on page 23 for information about the variables that can be used in expressions. The use of certain variables is dependent on the context of the expression.

Attributes

<booleanThreshold> has the following attribute:

Table 23. Attributes of the <booleanThreshold> element

Name	Description	Data type	Required?
expressionLanguage	Identifies the programming language in which the expression is written. Because the Java programming language is the only supported expression language, the only valid value for this attribute is `java`.	xsd:NMTOKEN	Yes

Contained within

<booleanThreshold> is contained within the following element:

- <thresholdRule>

Contains

<booleanThreshold> contains no elements.

Related concepts

"Expressions" on page 19
An expression is code that contains custom logic that can be added to a rule. Expressions can also access code that is external to the Active Correlation Technology engine. In the rule language, expressions are valid only within specific contexts, or rule language elements.

collectionRule element

The <collectionRule> element defines a rule according to the collection pattern.

Attributes

<collectionRule> has the following attributes:

Table 24. Attributes of the <collectionRule> element

Name	Description	Data type	Required?
name	Identifies the rule. This identifier must be unique within the rule block that contains this rule. It cannot contain a period.	xsd:NMTOKEN	Yes
processOnlyForwardedEvents	Determines whether the rule receives all events or only events that are forwarded from other rules. The default value is `false`, which indicates that the rule receives all events, including those that are forwarded from other rules.	xsd:boolean	No

Contained within

<collectionRule> is contained within the following element:

- <ruleBlock>

Contains

<collectionRule> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 25. Elements contained within the <collectionRule> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<activationInterval>	Optional. 0 or 1 occurrence is allowed.
<lifeCycleActions>	Optional. 0 or 1 occurrence is allowed.
<eventSelector>	Optional. 0 or 1 occurrence is allowed.
<groupingKey>	Optional. 0 or 1 occurrence is allowed.
<timeWindow>	Required. Only 1 occurrence is allowed.
<onTimeWindowComplete>	Optional. 0 or 1 occurrence is allowed.

Related concepts

"Collection pattern" on page 11
A collection rule is defined by the collection pattern. It collects a group of selected events within a time interval. It is a stateful rule.

comment element

The <comment> element can contain a description of the function and purpose of its containing rule set, rule block, rule, or variable.

Attributes

<comment> has no attributes.

Contained within

<comment> is contained within the following elements:

- <ruleSet>
- <ruleBlock>
- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>
- <timerRule>
- <variable>

Contains

<comment> contains no elements.

computationRule element

The <computationRule> element defines a rule according to the computation pattern.

Attributes

<computationRule> has the following attributes:

Table 26. Attributes of the <computationRule> element

Name	Description	Data type	Required?
name	Identifies the rule. This identifier must be unique within the rule block that contains this rule. It cannot contain a period.	xsd:NMTOKEN	Yes
processOnlyForwardedEvents	Determines whether the rule receives all events or only events that are forwarded from other rules. The default value is `false`, which indicates that the rule receives all events, including those that are forwarded from other rules.	xsd:boolean	No

Contained within

<computationRule> is contained within the following element:

- <ruleBlock>

Contains

<computationRule> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 27. Elements contained within the <computationRule> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<activationInterval>	Optional. 0 or 1 occurrence is allowed.
<lifeCycleActions>	Optional. 0 or 1 occurrence is allowed.
<eventSelector>	Optional. 0 or 1 occurrence is allowed.
<groupingKey>	Optional. 0 or 1 occurrence is allowed.
<computeFunction>	Required. Only 1 occurrence is allowed.
<timeWindow>	Required. Only 1 occurrence is allowed.
<onTimeWindowComplete>	Optional. 0 or 1 occurrence is allowed.

Related concepts

"Computation pattern" on page 11
A computation rule is defined by the computation pattern. It applies a computation (through an expression) to collected events as each event is received within a time interval. It is a stateful rule.

computedThreshold element

The <computedThreshold> element is valid only for the threshold rule. It contains an expression that is called as each event is received and that computes the threshold value based on the current event and on any other events that have matched the event selection criteria for the rule. The expression returns the computed threshold value to be stored in a variable that is defined for the rule. The rule then uses the computed threshold value to compare against the defined threshold value.

Details

Refer to "Variables" on page 23 for information about the variables that can be used in expressions. The use of certain variables is dependent on the context of the expression.

Attributes

<computedThreshold> has the following attributes:

Table 28. Attributes of the <computedThreshold> element

Name	Description	Data type	Required?
expressionLanguage	Identifies the programming language in which the expression is written. Because the Java programming language is the only supported expression language, the only valid value for this attribute is `java`.	xsd:NMTOKEN	Yes
threshold	Defines the threshold value that is to be met. This defined threshold value must be a string representation of a numeric value that can be converted to a data type that is valid for the rule variable.	xsd:string	Yes
assignTo	Identifies the name of the variable that holds the computed threshold value that is returned from this expression. This variable must already be defined for the rule (at the rule set, rule block, or rule level) using the <variable> element. It must be defined as one of the following numeric data types: • java.lang.Double • java.lang.Float • java.lang.Integer • java.lang.Long • java.lang.String If the variable is defined at the rule set or rule block level, it is not reinitialized after the rule pattern is matched.	xsd:NMTOKEN	Yes

Table 28. Attributes of the <computedThreshold> element (continued)

Name	Description	Data type	Required?
thresholdComparison	Defines the operator for comparing the computed threshold value to the defined threshold value. The valid values for this operator are: • `lessThan` • `lessThanOrEqualTo` • `greaterThan` • `greaterThanOrEqualTo`	xsd:string	Yes

Contained within

<computedThreshold> is contained within the following element:

- <thresholdRule>

Contains

<computedThreshold> contains no elements.

Related concepts

"Expressions" on page 19
An expression is code that contains custom logic that can be added to a rule. Expressions can also access code that is external to the Active Correlation Technology engine. In the rule language, expressions are valid only within specific contexts, or rule language elements.

computedValue element

The <computedValue> element contains an expression that runs when the rule receives an event to create a string value that is based on the value of one or more attributes of the event. This string value can then be used in the grouping key.

Details

Sometimes, a rule writer might want to use items such as these in the grouping key:

- A substring of an event attribute value. For example, if an event attribute value contains an embedded IP address, the expression within the <computedValue> element could extract that IP address as a unique value to use in the grouping key.
- Substrings of the values from several different event attributes. For example, the expression within the <computedValue> element could extract the substrings and combine them to create a unique value to use in the grouping key.

If the expression within the <computedValue> element returns a null value, the rule treats this null value as a missing attribute value.

Refer to "Variables" on page 23 for information about the variables that can be used in expressions. The use of certain variables is dependent on the context of the expression.

Attributes

<computedValue> has the following attribute:

Table 29. Attributes of the <computedValue> element

Name	Description	Data type	Required?
expressionLanguage	Identifies the programming language in which the expression is written. Because the Java programming language is the only supported expression language, the only valid value for this attribute is `java`.	xsd:NMTOKEN	Yes

Contained within

<computedValue> is contained within the following element:

- <groupingKey>

Contains

<computedValue> contains no elements.

Related concepts

"Expressions" on page 19
An expression is code that contains custom logic that can be added to a rule. Expressions can also access code that is external to the Active Correlation Technology engine. In the rule language, expressions are valid only within specific contexts, or rule language elements.

computeFunction element

The <computeFunction> element is valid only for the computation rule. It contains an expression that is called as each event is received and that returns a value to be stored in a variable that is defined for the rule. The value that is returned from this expression must match the data type of the variable that is named in the assignTo attribute of the <computeFunction> element.

Details

Refer to "Variables" on page 23 for information about the variables that can be used in expressions. The use of certain variables is dependent on the context of the expression.

Attributes

<computeFunction> has the following attributes:

Table 30. Attributes of the <computeFunction> element

Name	Description	Data type	Required?
expressionLanguage	Identifies the programming language in which the expression is written. Because the Java programming language is the only supported expression language, the only valid value for this attribute is `java`.	xsd:NMTOKEN	Yes

Table 30. Attributes of the <computeFunction> element (continued)

Name	Description	Data type	Required?
assignTo	Identifies the name of the variable that holds the value that is returned from this expression. This variable must already be defined for the rule (at the rule set, rule block, or rule level) using the <variable> element. If the variable is defined at the rule set or rule block level, it is not reinitialized after the rule pattern is matched.	xsd:NMTOKEN	Yes

Contained within

<computeFunction> is contained within the following element:

- <computationRule>

Contains

<computeFunction> contains no elements.

Related concepts

"Expressions" on page 19
An expression is code that contains custom logic that can be added to a rule. Expressions can also access code that is external to the Active Correlation Technology engine. In the rule language, expressions are valid only within specific contexts, or rule language elements.

dateTime element

The <dateTime> element specifies the date and time when a rule is activated or deactivated. However, the rule is activated or deactivated only if the rule has been loaded into a running Active Correlation Technology engine prior to that specified time.

Details

If the rule has not been loaded into a running Active Correlation Technology engine prior to the specified time for activation, the rule is never activated. If the rule has not been loaded into a running Active Correlation Technology engine prior to the specified time for deactivation, the rule is set to the state that is defined by the <start> element, and it is never deactivated by the <stop> element.

The content of the <dateTime> element must be a string that follows the format for the `dateTime` data type in the standard XML schema. For example, `dateTime` consists of finite-length sequences of characters in the following form:

yyyy '-' *mm* '-' *dd* 'T' *hh* ':' *mm* ':' *ss* ('.' *s*+)? (*zzzzzz*)?

- *yyyy* is a four-or-more digit numeral that represents the year. If it is more than four digits, leading zeros are prohibited, and `0000` is prohibited.
- The remaining '-'s are separators between parts of the date portion.
- The first *mm* is a two-digit numeral that represents the month, starting with `01`.
- *dd* is a two-digit numeral that represents the day of the month, starting with `01`.
- `T` is a separator that indicates that the time-of-day follows.

- *hh* is a two-digit numeral that represents the hour of the day in the 24-hour system, starting with `00` and ending with `23`.
- `:` is a separator between parts of the time-of-day portion.
- The second *mm* is a two-digit numeral that represents the minute, starting with `00` and ending with `59`.
- *ss* is a two-digit numeral that represents the whole seconds, starting with `00` and ending with `59`.
- `'.'` *s*+, if present, represents the fractional seconds.
- *zzzzzz*, if present, represents the time zone. The time zone consists of finite-length sequences of characters in the form `(('+' | '-')` *hh* `':'` *mm*`) | 'Z'`, where:
  - `'+'`, if present, represents a nonnegative duration, and `'-'` must not be present.
  - `'-'`, if present, represents a nonpositive duration, and `'+'` must not be present.
  - *hh* is a two-digit numeral that represents the hours, starting with `00` and ending with `14`.
  - *mm* is a two-digit numeral that represents the minutes, starting with `00` and ending with `59`. However, if the hours value is `14`, the minutes value must be `00`
  - Z is shorthand for UTC (either `+00:00` or `-00:00`), and as such, no other time zone elements must be present.

Here are two examples of the content of the <dateTime> element:

- `2005-06-01T13:05:06.07` is June 1, 2005 at 6 seconds, and 7 hundredths of a second past 1:05 p.m. local time.
- `2005-06-01T13:05:06.07Z` is June 1, 2005 at 6 seconds, and 7 hundredths of a second past 1:05 p.m. UTC time, which would be June 1, 2005 at 6 seconds, and 7 hundredths of a second past 9:05 a.m. EDT (or `2005-06-01T09:05:06.07-04:00`)

Attributes

<dateTime> has no attributes.

Contained within

<dateTime> is contained within the following elements:

- <start>
- <stop>

Contains

<dateTime> contains no elements.

deactivateOnEvent element

The <deactivateOnEvent> element defines the events that can deactivate the rule or, for rules that are defined with a <groupingKey> element, a rule instance.

Following are the three possible ways of selecting events:

- The use of one or more <eventType> elements with a <filteringPredicate> element

- The use of one or more <eventType> elements without a <filteringPredicate> element
- The use of a <filteringPredicate> element without any <eventType> elements

If the rule is active and no <eventType> or <filteringPredicate> element is coded, any event that occurs is selected.

Not coding any <eventType> elements can negatively impact system performance.

Assume that you want to select all events of type `Audit Failure`. You can use a filtering predicate to further refine the selection criteria to include only the events that have an event attribute with a certain value. For example, you would code an <eventType> element to select all events of type `Audit Failure`, and code a <filteringPredicate> element to select only those events that have a hostname attribute with the value `MyCriticalSystem`.

Attributes

<deactivateOnEvent> has no attributes.

Contained within

<deactivateOnEvent> is contained within the following elements:

- <activationInterval>
- <activationByGroupingKey>

Contains

<deactivateOnEvent> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 31. Elements contained within the <deactivateOnEvent> element

Element	Required or optional?
<eventType>	Optional. 0 or more occurrences are allowed.
<filteringPredicate>	Optional. 0 or 1 occurrence is allowed.

duplicateRule element

The <duplicateRule> element defines a rule according to the duplicate pattern.

Attributes

<duplicateRule> has the following attributes:

Table 32. Attributes of the <duplicateRule> element

Name	Description	Data type	Required?
name	Identifies the rule. This identifier must be unique within the rule block that contains this rule. It cannot contain a period.	xsd:NMTOKEN	Yes
processOnlyForwardedEvents	Determines whether the rule receives all events or only events that are forwarded from other rules. The default value is `false`, which indicates that the rule receives all events, including those that are forwarded from other rules.	xsd:boolean	No

Contained within

<duplicateRule> is contained within the following element:

- <ruleBlock>

Contains

<duplicateRule> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 33. Elements contained within the <duplicateRule> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<activationInterval>	Optional. 0 or 1 occurrence is allowed.
<lifeCycleActions>	Optional. 0 or 1 occurrence is allowed.
<eventSelector>	Optional. 0 or 1 occurrence is allowed.
<groupingKey>	Optional. 0 or 1 occurrence is allowed.
<timeWindow>	Required. Only 1 occurrence is allowed.
<onDetection>	Optional. 0 or 1 occurrence is allowed.
<onNextEvent>	Optional. 0 or 1 occurrence is allowed.
<onTimeWindowComplete>	Optional. 0 or 1 occurrence is allowed.

Related concepts

"Duplicate pattern" on page 12
A duplicate rule is defined by the duplicate pattern. It counts the second and subsequent events that are accepted within the specified time interval but skips the rule set processing for these events. It is a stateful rule.

eventAttribute element

The <eventAttribute> element provides a way to associate an event type and an event attribute as part of the attribute alias name that is defined by the <attributeAlias> element.

Attributes

<eventAttribute> has the following attributes:

Table 34. Attributes of the <eventAttribute> element

Name	Description	Data type	Required?
type	Defines the name of the type of event. This is the same name that is used for the type attribute on the <eventType> element.	xsd:NMTOKEN	Yes
attributeName	Specifies the fully qualified name of the event attribute that is being associated with other event attributes through the attribute alias name. This name must match the name that is used in the act_event variable to call the getAttribute method.	xsd:string	Yes

Contained within

<eventAttribute> is contained within the following element:

- <attributeAlias>

Contains

<eventAttribute> contains no elements.

eventCountThreshold element

The <eventCountThreshold> element is valid only for the threshold rule. It defines the number of events that must meet the event selection criteria in a certain time period. The <eventCountThreshold> element also specifies one of two possible time interval modes, either fixed or sliding, for the time window.

Details

fixed interval

A fixed interval begins when the first event that meets the event selection criteria is received and ends when one of the following occurs:

- The rule meets its threshold within the specified time duration.
- The specified time duration has passed.

sliding interval

A sliding interval begins when the first event that meets the event selection criteria is received. However, when the rule has not met its threshold and the specified time duration has passed, the time window adjusts (slides) the beginning time to the event reception time for a new "first" event, which is typically the next event that is accepted. The sliding interval continues to adjust in this way until one of the following occurs:

- The rule meets its threshold within the specified time duration.
- After the event that begins the time window is received, no subsequent events are received within the specified time duration.

The event that begins the time window (becomes the new "first" event) is the event with a reception time that meets this criteria: the reception time, added to the time interval duration for the rule, is greater than the current time. Here is the criteria in the form of an equation:

event reception time + time interval duration for rule > current time

When no such event exists, the sliding interval cannot adjust the time any further, and the interval ends.

The threshold rule counts each accepted event until either the threshold is met or the time period ends. Then it runs the actions that are defined within the <onDetection> element or the <onTimeOut> element, as appropriate.

<onDetection> actions
These actions run when the event count equals the value that is defined by the threshold attribute of the <eventCountThreshold> element, which indicates that the threshold is met.

<onTimeOut> actions
When these actions run is dependent on whether the time interval mode is fixed or sliding.

fixed mode
With fixed mode, these actions run when the time window expires.

sliding mode
With sliding mode, these actions run if, after the event that begins the time window is received, no subsequent events are received within the specified time duration. In other words, no event is received with a reception time that, when added to the time interval duration for the rule, is greater than the current time.

The time interval mode for the time window is defined by the timeIntervalMode attribute of the <eventCountThreshold> element. The following scenario illustrates the behavior of, and differences between, the two possible time interval modes.

Scenario that illustrates fixed and sliding modes

Assume that the rule receives four events that meet the event selection criteria, one event at each of these times: 8:00, 8:04, 8:06, and 8:07. The event count threshold is 3, and the duration for the time window is 5 minutes.

Rule behavior with `fixed` mode
With this time interval mode, the threshold rule begins processing at 8:00, and it runs the <onTimeOut> actions at 8:05 because it receives only 2 events in 5 minutes. Therefore, it does not meet its threshold within the time window. When the third event is received at 8:06, the threshold rule begins processing again, and it runs the <onTimeOut> actions at 8:11 because it receives only 2 events in 5 minutes.

The fixed mode is static.

Rule behavior with `sliding` mode
With this time interval mode, the threshold rule begins processing at 8:00. At 8:05, when the time window is scheduled to complete, the rule determines that it has received only 2 events. The rule then discards the

event that it received at 8:00 and recalculates the duration to end at 8:09 (since the first event is now the one that it received at 8:04). When the rule receives the event at 8:07, it runs the <onDetection> actions because it has now met its threshold (3 events, at 8:04, 8:06, and 8:07) within the latest time window (8:04 – 8:09).

The sliding mode is dynamic in that it continues to adjust (to slide) the beginning time in an attempt to meet its threshold within the time window.

Now assume that the rule receives 4 events that meet the event selection criteria, one event at each of these times: 8:00, 8:04, 8:06, and 8:10. The event count threshold is 3, and the duration for the time window is 5 minutes.

Rule behavior with `sliding` mode

In this case, the threshold rule begins processing at 8:00. At 8:05, when the time window is scheduled to complete, the rule determines that it has received only 2 events. The rule then discards the event that it received at 8:00 and recalculates the duration to end at 8:09 (since the first event is now the one that it received at 8:04).

At 8:09, when the time window is now scheduled to complete, the rule determines that it has received only 2 events. The rule then discards the event that it received at 8:04 and recalculates the duration to end at 8:11 (since the first event is now the one that it received at 8:06).

At 8:11, when the time window is now scheduled to complete, the rule determines that it has received only 2 events. The rule then discards the event that it received at 8:06 and recalculates the duration to end at 8:15 (since the first event is now the one that it received at 8:10).

At 8:15, when the time window is now scheduled to complete, the rule determines that it has received no events since the event at 8:10 that began the time window. The rule then runs the <onTimeOut> actions.

Attributes

<eventCountThreshold> has the following attributes:

Table 35. Attributes of the <eventCountThreshold> element

Name	Description	Data type	Required?
threshold	Defines the number of events that must meet the event selection criteria within a certain time period. This is the event count threshold that is to be met. This value must be a positive integer.	xsd:positiveInteger	Yes
timeIntervalMode	Defines whether the time interval for the time window is fixed or sliding. The valid values for this attribute are: • `fixed` (the default value) • `sliding`	xsd:string	No

Contained within

<eventCountThreshold> is contained within the following element:

- <thresholdRule>

Contains

<eventCountThreshold> contains no elements.

eventSelector element

The <eventSelector> element defines the events that are selected for processing by a rule.

Details

Following are the three possible ways of selecting events:

- The use of one or more <eventType> elements with a <filteringPredicate> element
- The use of one or more <eventType> elements without a <filteringPredicate> element
- The use of a <filteringPredicate> element without any <eventType> elements

In special cases where you want a rule to process all events, you have the following options:

- Do not code an <eventSelector> element.
- Code an <eventSelector> element that contains no elements.

Not coding any <eventType> elements can negatively impact system performance.

Assume that you want to select all events of type `Audit Failure`. You can use a filtering predicate to further refine the selection criteria to include only the events that have an event attribute with a certain value. For example, you would code an <eventType> element to select all events of type `Audit Failure`, and code a <filteringPredicate> element to select only those events that have a hostname attribute with the value `MyCriticalSystem`.

Attributes

<eventSelector> has the following attribute:

Table 36. Attributes of the <eventSelector> element

Name	Description	Data type	Required?
alias	This attribute is valid only within a sequence rule, which is the only rule that has multiple <eventSelector> elements. It uniquely names an event that is selected by a certain event selector in the sequence rule. Filtering predicates and actions can then use this alias name to access that event.	xsd:NMTOKEN	No

Contained within

<eventSelector> is contained within the following elements:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>

Contains

<eventSelector> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 37. Elements contained within the <eventSelector> element

Element	Required or optional?
<eventType>	Optional. 0 or more occurrences are allowed.
<filteringPredicate>	Optional. 0 or 1 occurrence is allowed.

eventType element

The <eventType> element defines the type of event that is selected for processing by a rule or that activates or deactivates the rule.

Attributes

<eventType> has the following attribute:

Table 38. Attributes of the <eventType> element

Name	Description	Data type	Required?
type	Defines the event type. For events that conform to the Common Base Event specification, this name is the value of the extensionName attribute. For IBM Tivoli Enterprise Console® events, this name is the event class name that is defined in the BAROC file. Events that are based on other formats might use a different attribute to specify the event type.	xsd:NMTOKEN	Yes

Contained within

<eventType> is contained within the following elements:

- <activateOnEvent>
- <deactivateOnEvent>

- <eventSelector>

Contains

<eventType> contains no elements.

filteringPredicate element

The <filteringPredicate> element contains an expression that further restricts which events are selected for processing by the rule or are selected to activate or deactivate the rule. Therefore, events can be filtered even more comprehensively than filtering only by event type through the <eventType> element.

Details

The expression defines a condition and returns a boolean value, either `true` (the condition is met) or `false` (the condition is not met).

Refer to "Variables" on page 23 for information about the variables that can be used in expressions. The use of certain variables is dependent on the context of the expression.

Attributes

<filteringPredicate> has the following attribute:

Table 39. Attributes of the <filteringPredicate> element

Name	Description	Data type	Required?
expressionLanguage	Identifies the programming language in which the expression is written. Because the Java programming language is the only supported expression language, the only valid value for this attribute is `java`.	xsd:NMTOKEN	Yes

Contained within

<filteringPredicate> is contained within the following elements:

- <activateOnEvent>
- <deactivateOnEvent>
- <eventSelector>

Contains

<filteringPredicate> contains no elements.

Related concepts

"Expressions" on page 19
An expression is code that contains custom logic that can be added to a rule. Expressions can also access code that is external to the Active Correlation Technology engine. In the rule language, expressions are valid only within specific contexts, or rule language elements.

filterRule element

The <filterRule> element defines a rule according to the filter pattern.

Attributes

<filterRule> has the following attributes:

Table 40. Attributes of the <filterRule> element

Name	Description	Data type	Required?
name	Identifies the rule. This identifier must be unique within the rule block that contains this rule. It cannot contain a period.	xsd:NMTOKEN	Yes
processOnlyForwardedEvents	Determines whether the rule receives all events or only events that are forwarded from other rules. The default value is `false`, which indicates that the rule receives all events, including those that are forwarded from other rules.	xsd:boolean	No

Contained within

<filterRule> is contained within the following element:

- <ruleBlock>

Contains

<filterRule> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 41. Elements contained within the <filterRule> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<activationInterval>	Optional. 0 or 1 occurrence is allowed.
<lifeCycleActions>	Optional. 0 or 1 occurrence is allowed.
<eventSelector>	Optional. 0 or 1 occurrence is allowed.
<onDetection>	Optional. 0 or 1 occurrence is allowed.

Related concepts

"Filter pattern" on page 13
A filter rule is defined by the filter pattern. It takes a certain action when it accepts an event. It acts only on a single event and is therefore a stateless rule.

groupingKey element

Typically, each active rule has one rule instance, or copy, that is running in the Active Correlation Technology engine. However, sometimes the same rule is needed for different groups of events, which are often related to different groups of resources. The grouping key is one or more event attributes, or parts of event attributes, that can be used to separate the selected events into different groups for unique processing as a group. The <groupingKey> element defines the grouping key for a rule. The purpose of the <groupingKey> element is to direct the rule to create a separate rule instance (or copy of itself) for each group of events that share common characteristics (as defined by the values of their attributes that comprise the grouping key).

Details

The following two scenarios illustrate the importance of the grouping key.

Scenario 1:
Two events occur, a DB2down event and a DB2up event. The DB2 program runs on three computers that are named A, B, and C. A sequence rule is defined to correlate a DB2down event with a DB2up event and to alert the operator when the DB2 program stops and does not restart.

If the sequence rule is defined without the grouping key, and a DB2down event is received from computer A, a DB2up event from any of the computers would complete the sequence, but this would not accomplish what was intended. However, if the grouping key were defined as the hostname attribute, a unique copy, or instance, of the rule would be created for each unique value of the hostname attribute in the selected events. The Active Correlation Technology engine would send each event to the correct rule instance (the rule instance for that event's hostname value). Therefore, if a DB2down event is received from computer A, the Active Correlation Technology engine would create a rule instance for computer A. If a DB2down event is received from computer B, the Active Correlation Technology engine would create a second rule instance for computer B. When a DB2up event is received from computer B, the Active Correlation Technology engine processes that event in the second rule instance. The sequence is complete, and the operator is alerted because the DB2down and DB2up events from computer B are correctly correlated.

Scenario 2:
An event for an `Incorrect login attempted` message occurs on all computers in a particular environment. The event contains a user ID. A threshold rule is defined to issue a warning to the operator if this event occurs more than 10 times in 5 minutes.

A grouping key could be defined as the user ID. Then, a new rule instance would be created for each unique user ID. Each user's login attempts would be tracked in a unique threshold rule instance, with each instance having a separate count of the number of login attempts by that user. The operator would receive a warning if any user ID exceeds 10 incorrect logins in 5 minutes.

Other variations of this idea include:

- The grouping key could be defined as the host name rather than the user ID. This option could detect a large number of incorrect login attempts on a single computer.

- The grouping key could be defined as a combination of the host name and user ID. This option could detect a potential hack attempt by a specific user ID against a specific computer.

If the same attribute is in all event types that are specified for a rule, the use of the <attributeName> element is the simplest and most common way to define a grouping key.

Attributes

<groupingKey> has the following attribute:

Table 42. Attributes of the <groupingKey> element

Name	Description	Data type	Required?
missingAttributeHandling	Defines the action that the rule must take under either of these conditions: • When a selected event has an attribute that participates in the grouping key but the value for that attribute is missing • When the expression within the <computedValue> element returns a null value. The rule treats this null value as a missing attribute value. The valid values for the missingAttributeHandling attribute are: • `ignoreEvent` (the default value), which means that the rule ignores the event and takes no action on it. • `ignoreAttribute`, which means that the rule accepts the event but ignores the attribute with the missing value. The Active Correlation Technology engine then includes a substitute value for the attribute.	xsd:string	No

Contained within

<groupingKey> is contained within the following elements:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <sequenceRule>
- <thresholdRule>

Contains

<groupingKey> contains the following elements.

Table 43. Elements contained within the <groupingKey> element

Element	Required or optional?
<attributeAlias>	1 of these elements is required. Coding more than 1 of these elements is optional. Multiple occurrences of all three elements are allowed. These elements can be coded in any order.
<attributeName>	
<computedValue>	

import element

The <import> element contains an expression that specifies the external modules (such as Java classes) to import for use in other expressions within rules.

Details

The expression code is a string within the <import> element. The Active Correlation Technology compiler uses the import statements that are provided by the <import> elements to compile expression code within rules that call external methods.

Attributes

<import> has the following attribute:

Table 44. Attributes of the <import> element

Name	Description	Data type	Required?
expressionLanguage	Identifies the programming language in which the expression is written. Because the Java programming language is the only supported expression language, the only valid value for this attribute is `java`.	xsd:NMTOKEN	Yes

Contained within

<import> is contained within the following elements:

- <ruleSet>
- <ruleBlock>

Contains

<import> contains no elements.

Related concepts

"Importing and accessing external modules and objects" on page 20
This example indicates how you can make external code (such as Java classes) and external objects accessible to expressions. An external object is an object that an application creates to communicate with expressions.

inactiveWhenLoaded element

The <inactiveWhenLoaded> element specifies that a rule is inactive when it is loaded by the Active Correlation Technology engine. The rule remains inactive until it is activated by another means.

Attributes

<inactiveWhenLoaded> has no attributes.

Contained within

<inactiveWhenLoaded> is contained within the following element:

- <start>

Contains

<inactiveWhenLoaded> contains no elements.

lifeCycleActions element

The <lifeCycleActions> element contains elements that define the actions to take at the four primary stages in the life cycle of a rule.

Details

The actions that are defined for the load and activation stages are called after the rule is actually loaded or activated but before the rule begins any processing. The actions that are defined for the deactivation and unload stages are called immediately before the rule is actually deactivated or unloaded.

Attributes

<lifeCycleActions> has no attributes.

Contained within

<lifeCycleActions> is contained within the following elements:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>
- <timerRule>

Contains

<lifeCycleActions> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 45. Elements contained within the <lifeCycleActions> element

Element	Required or optional?
<onLoad>	Optional. 0 or 1 occurrence is allowed.
<onActivation>	Optional. 0 or 1 occurrence is allowed.
<onDeactivation>	Optional. 0 or 1 occurrence is allowed.
<onUnload>	Optional. 0 or 1 occurrence is allowed.

never element

The <never> element specifies that a rule is never deactivated at a certain time. A rule can still be deactivated by an event or other means.

Attributes

<never> has no attributes.

Contained within

<never> is contained within the following element:

- <stop>

Contains

<never> contains no elements.

onActivation element

The <onActivation> element specifies the action or set of actions to take when the rule is activated. The <onActivation> action is called after the rule is activated but before the rule begins any processing.

Details

If the rule set contains multiple rules that are activated at the same date and time, or by the same event, and that have the same time window, the following actions for these rules do not run at exactly the same time:

- Rule response actions within the <onTimeOut> and <onTimeWindowComplete> elements
- Life cycle actions within the <onActivation> and <onDeactivation> elements

These actions run sequentially in any order. They do not necessarily run in the order in which they are coded in the rule set. Because each action must be completed before the next action in the sequence begins, the actions do not run at the same time.

Attributes

<onActivation> has no attributes.

Contained within

<onActivation> is contained within the following element:

- <lifeCycleActions>

Contains

<onActivation> contains the following element:

Table 46. Elements contained within the <onActivation> element

Element	Required or optional?
<action>	Optional. 0 or more occurrences are allowed.

onDeactivation element

The <onDeactivation> element specifies the action or set of actions to take when the rule is deactivated. The <onDeactivation> action is called immediately before the rule is deactivated.

Details

If the rule set contains multiple rules that are activated at the same date and time, or by the same event, and that have the same time window, the following actions for these rules do not run at exactly the same time:

- Rule response actions within the <onTimeOut> and <onTimeWindowComplete> elements
- Life cycle actions within the <onActivation> and <onDeactivation> elements

These actions run sequentially in any order. They do not necessarily run in the order in which they are coded in the rule set. Because each action must be completed before the next action in the sequence begins, the actions do not run at the same time.

Attributes

<onDeactivation> has no attributes.

Contained within

<onDeactivation> is contained within the following element:

- <lifeCycleActions>

Contains

<onDeactivation> contains the following element:

Table 47. Elements contained within the <onDeactivation> element

Element	Required or optional?
<action>	Optional. 0 or more occurrences are allowed.

onDetection element

The <onDetection> element is valid only for the duplicate, filter, sequence, and threshold rules. It specifies the action or set of actions to take when the rule pattern is detected.

Details

Table 48 describes how the rule pattern is detected for each rule type where an <onDetection> action is valid.

Table 48. How a rule pattern is detected based on the rule type

Rule type	How the rule pattern is detected
duplicate	This rule pattern is detected when the first event that meets the event selection criteria is received.
filter	This rule pattern is detected when any event that meets the event selection criteria is received.
sequence	This rule pattern is detected when a sequence of events that meet the event selection criteria are received in the appropriate order and within the time window.
threshold	This rule pattern is detected when events that meet the event selection criteria are received within the time window and the threshold is met.

Attributes

<onDetection> has no attributes.

Contained within

<onDetection> is contained within the following elements:

- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>

Contains

<onDetection> contains the following element:

Table 49. Elements contained within the <onDetection> element

Element	Required or optional?
<action>	Optional. 0 or more occurrences are allowed.

onLoad element

The <onLoad> element specifies the action or set of actions to take when the rule is loaded (or deployed) in the running Active Correlation Technology engine. The <onLoad> action is called after the rule is loaded but before the rule begins any processing.

Attributes

<onLoad> has no attributes.

Contained within

<onLoad> is contained within the following element:

- <lifeCycleActions>

Contains

<onLoad> contains this element:

Table 50. Elements contained within the <onLoad> element

Element	Required or optional?
<action>	Optional. 0 or more occurrences are allowed.

onNextEvent element

The <onNextEvent> element is valid only for the duplicate rule. It specifies the action or set of actions to take when the duplicate rule receives the second and each subsequent event that meets the event selection criteria within the specified time window.

Details

For duplicate rules, the Active Correlation Technology engine skips the rule set processing for the second and each subsequent event that matches the event selection criteria within the specified time window. Therefore, the only reason for coding an <onNextEvent> action is to specify alternative processing for the second and each subsequent event.

Attributes

<onNextEvent> has no attributes.

Contained within

<onNextEvent> is contained within the following element:

- <duplicateRule>

Contains

<onNextEvent> contains the following element:

Table 51. Elements contained within the <onNextEvent> element

Element	Required or optional?
<action>	Optional. 0 or more occurrences are allowed.

onTimeOut element

The <onTimeOut> element is valid only for the sequence and threshold rules. It specifies the action or set of actions to take if the time window for the rule expires.

Details

Table 52 on page 96 describes how the time window expires for each rule type where an <onTimeOut> action is valid.

Table 52. How the time window expires based on the rule type

Rule type	How the time window expires
sequence	The time window expires if one or more events are accepted but the full sequence of events is not received within the time window.
threshold	The time window expires if one or more events are accepted but the threshold is not met within the time window.

If the rule set contains multiple rules that are activated at the same date and time, or by the same event, and that have the same time window, the following actions for these rules do not run at exactly the same time:

- Rule response actions within the <onTimeOut> and <onTimeWindowComplete> elements
- Life cycle actions within the <onActivation> and <onDeactivation> elements

These actions run sequentially in any order. They do not necessarily run in the order in which they are coded in the rule set. Because each action must be completed before the next action in the sequence begins, the actions do not run at the same time.

Attributes

<onTimeOut> has no attributes.

Contained within

<onTimeOut> is contained within the following elements:

- <sequenceRule>
- <thresholdRule>

Contains

<onTimeOut> contains the following element:

Table 53. Elements contained within the <onTimeOut> element

Element	Required or optional?
<action>	Optional. 0 or more occurrences are allowed.

onTimeWindowComplete element

The <onTimeWindowComplete> element is valid only for the collection, computation, duplicate, and timer rules. It specifies the action or set of actions to take when the time window for the rule is over.

Details

If the rule set contains multiple rules that are activated at the same date and time, or by the same event, and that have the same time window, the following actions for these rules do not run at exactly the same time:

- Rule response actions within the <onTimeOut> and <onTimeWindowComplete> elements
- Life cycle actions within the <onActivation> and <onDeactivation> elements

These actions run sequentially in any order. They do not necessarily run in the order in which they are coded in the rule set. Because each action must be completed before the next action in the sequence begins, the actions do not run at the same time.

Attributes

<onTimeWindowComplete> has no attributes.

Contained within

<onTimeWindowComplete> is contained within the following elements:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <timerRule>

Contains

<onTimeWindowComplete> contains the following element:

Table 54. Elements contained within the <onTimeWindowComplete> element

Element	Required or optional?
<action>	Optional. 0 or more occurrences are allowed.

onUnload element

The <onUnload> element specifies the action or set of actions to take when the rule is unloaded, or removed, from the running Active Correlation Technology engine. The <onUnload> action is called immediately before the rule is unloaded.

Attributes

<onUnload> has no attributes.

Contained within

<onUnload> is contained within the following element:

- <lifeCycleActions>

Contains

<onUnload> contains the following element:

Table 55. Elements contained within the <onUnload> element

Element	Required or optional?
<action>	Optional. 0 or more occurrences are allowed.

ruleBlock element

The <ruleBlock> element provides the way to group related rules and organize rules in a hierarchy.

Attributes

<ruleBlock> has the following attribute:

Table 56. Attributes of the <ruleBlock> element

Name	Description	Data type	Required?
name	Identifies the rule block. This identifier must be unique within the rule set or rule block that contains this rule block. It cannot contain a period.	xsd:NMTOKEN	Yes

Contained within

<ruleBlock> is contained within the following elements:

- <ruleSet>
- <ruleBlock>

Contains

<ruleBlock> contains the following elements.

If they are coded, the <comment>, <import>, and <variable> elements must be coded in the order that is shown. The remaining elements can be coded in any order.

Table 57. Elements contained within the <ruleBlock> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<import>	Optional. 0 or more occurrences are allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<ruleBlock>	Optional. 0 or more occurrences are allowed.
<collectionRule>	Optional. 0 or more occurrences are allowed.
<computationRule>	Optional. 0 or more occurrences are allowed.
<duplicateRule>	Optional. 0 or more occurrences are allowed.
<filterRule>	Optional. 0 or more occurrences are allowed.
<sequenceRule>	Optional. 0 or more occurrences are allowed.
<thresholdRule>	Optional. 0 or more occurrences are allowed.
<timerRule>	Optional. 0 or more occurrences are allowed.

ruleSet element

The <ruleSet> element that is defined by `act:ruleSet` is the root element for the Active Correlation Technology rule language. All other elements are contained within this <ruleSet> element.

Details

The <ruleSet> elements that are defined by the Active Correlation Technology language schema (`act:ruleSet`) and by the Active Correlation Technology base rule set schema (`br:ruleSet`) are duplicates. However, when you create a rule set, you

must specify the following namespace on the <ruleSet> element: `act:ruleSet`.

Attributes

<ruleSet> has the following attribute:

Table 58. Attributes of the <ruleSet> element

Name	Description	Data type	Required?
name	Identifies the rule set. This identifier must be unique. It cannot contain a period.	xsd:NMTOKEN	Yes

Contained within

Because <ruleSet> is the root element for the rule language, it is not contained within any element.

Contains

<ruleSet> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 59. Elements contained within the <ruleSet> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<import>	Optional. 0 or more occurrences are allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<ruleBlock>	Optional. 0 or more occurrences are allowed.

runUntilDeactivated element

The <runUntilDeactivated> element specifies that the time window continues to be open until the rule is deactivated. Therefore, the time window for this rule starts when the rule begins processing, and it does not stop until the rule is deactivated or removed from the rule set, or the Active Correlation Technology engine shuts down.

Details

The specific behavior of a rule that includes the <runUntilDeactivated> element is dependent on the rule type. Table 60 on page 100 describes the rule behavior for each rule type where the <timeWindow> element is valid and contains the <runUntilDeactivated> element.

Table 60. Rule behavior when <runUntilDeactivated> is coded

Rule type	Rule behavior when <runUntilDeactivated> is coded
collection	The collection rule accepts the first event that meets its event selection criteria, and it continues to accept and process events until the rule is deactivated, at which time the actions that are defined within the <onTimeWindowComplete> element are run, immediately followed by the actions that are defined within the <onDeactivation> element.
computation	The computation rule accepts the first event that meets its event selection criteria, and it continues to accept and process events until the rule is deactivated, at which time the actions that are defined within the <onTimeWindowComplete> element are run, immediately followed by the actions that are defined within the <onDeactivation> element.
duplicate	The duplicate rule accepts the first event that meets its event selection criteria, and it continues to accept and process events until the rule is deactivated, at which time the actions that are defined within the <onTimeWindowComplete> element are run, immediately followed by the actions that are defined within the <onDeactivation> element.
sequence	The sequence rule accepts the first event that meets its event selection criteria, and it continues to accept and process events until either of the following circumstances occurs: • The sequence pattern is detected. When this occurs, the actions that are defined within the <onDetection> element are run, and the rule returns to its initial state. Event processing by this rule begins again, and this process can repeat many times until the rule is deactivated. • The rule is deactivated while it is processing events. When this occurs, the actions that are defined within the <onTimeOut> element are run, immediately followed by the actions that are defined within the <onDeactivation> element.
threshold	The threshold rule accepts the first event that meets its event selection criteria, and it continues to accept and process events until either of the following circumstances occurs: • The threshold pattern is detected. When this occurs, the actions that are defined within the <onDetection> element are run, and the rule returns to its initial state. Event processing by this rule begins again, and this process can repeat many times until the rule is deactivated. • The rule is deactivated while it is processing events. When this occurs, the actions that are defined within the <onTimeOut> element are run, immediately followed by the actions that are defined within the <onDeactivation> element.
timer	After the timer rule becomes active, it does nothing until it is deactivated, at which time the actions that are defined within the <onTimeWindowComplete> element are run, immediately followed by the actions that are defined within the <onDeactivation> element. The repeat attribute on the <timerRule> element is ignored.

Attributes

<runUntilDeactivated> has no attributes.

Contained within

<runUntilDeactivated> is contained within the following element:

- <timeWindow>

Contains

<runUntilDeactivated> contains no elements.

sequenceRule element

The <sequenceRule> element defines a rule according to the sequence pattern. The sequence rule is the only rule that allows multiple event selectors. It also requires a minimum of two event selectors.

Attributes

<sequenceRule> has the following attributes:

Table 61. Attributes of the <sequenceRule> element

Name	Description	Data type	Required?
name	Identifies the rule. This identifier must be unique within the rule block that contains this rule. It cannot contain a period.	xsd:NMTOKEN	Yes
processOnlyForwardedEvents	Determines whether the rule receives all events or only events that are forwarded from other rules. The default value is `false`, which indicates that the rule receives all events, including those that are forwarded from other rules.	xsd:boolean	No
arrivalOrder	Defines whether the events must arrive in the order in which the <eventSelector> elements are coded for the rule. The valid values are: • `inOrder` (the default value) • `randomOrder`	xsd:string	No

If the value of the arrivalOrder attribute is `randomOrder`, the order in which the <eventSelector> elements are coded is important. The <eventSelector> elements with the most specific event selection criteria should be coded before the <eventSelector> elements with less specific event selection criteria. Otherwise, the sequence is not detected when it should be.

For example, assume the following circumstances:

- Three <eventSelector> elements are defined.
- The first <eventSelector> element is checking for the event `eventA`.
- The second <eventSelector> element is checking for any event.
- The third <eventSelector> element is checking for the event `eventB`.
- The following events are presented to the system within the specified time window: `eventA`, `eventB`, `eventC`.

The rule behavior is as follows, and the sequence is not detected when it should be:

1. The first event, `eventA`, is accepted by the first <eventSelector> element.
2. The second event, `eventB`, is accepted by the second <eventSelector> element.
3. The third event, `eventC`, is ignored.

Assume the following circumstances, where the <eventSelector> elements are coded correctly, with the most specific event selection criteria preceding the less specific event selection criteria:

- The first <eventSelector> element is checking for the event `eventA`.
- The second <eventSelector> element is checking for the event `eventB`.
- The third <eventSelector> element is checking for any event.

The rule behavior is as follows, and the sequence is detected:

1. The first event, `eventA`, is accepted by the first <eventSelector> element.
2. The second event, `eventB`, is accepted by the second <eventSelector> element.
3. The third event, `eventC`, is accepted by the third <eventSelector> element.

Contained within

<sequenceRule> is contained within the following element:

- <ruleBlock>

Contains

<sequenceRule> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 62. Elements contained within the <sequenceRule> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<activationInterval>	Optional. 0 or 1 occurrence is allowed.
<lifeCycleActions>	Optional. 0 or 1 occurrence is allowed.
<eventSelector>	2 occurrences of this element are required for the sequence rule. Additional occurrences are allowed.
<groupingKey>	Optional. 0 or 1 occurrence is allowed.
<timeWindow>	Required. Only 1 occurrence is allowed.
<onDetection>	Optional. 0 or 1 occurrence is allowed.
<onTimeOut>	Optional. 0 or 1 occurrence is allowed.

Related concepts

"Sequence pattern" on page 13
A sequence rule is defined by the sequence pattern. It detects whether a certain sequence of events arrives within a time interval. The sequence can be ordered or random. A sequence rule is a stateful rule.

start element

The <start> element defines whether a rule is activated at a certain time on a certain date or is activated when the rule is loaded by the Active Correlation Technology engine.

Details

If the <start> element is not coded at all, the default start time is the same as that defined by the <whenLoaded> element.

Attributes

<start> has no attributes.

Contained within

<start> is contained within the following element:

- <activationTime>

Contains

<start> contains the following elements:

Table 63. Elements contained within the <start> element

Element	Required or optional?
<dateTime>	1 of these elements is required, and only 1 occurrence of the chosen element is allowed.
<whenLoaded>	
<inactiveWhenLoaded>	

stop element

The <stop> element defines whether a rule is deactivated at a certain time on a certain date, after a certain duration, or never at a certain time.

Details

If the <stop> element is not coded at all, the default stop time is the same as that defined by the <never> element.

Attributes

The <stop> element has no attributes.

Contained within

The <stop> element is contained within the following element:

- <activationTime>

Contains

The <stop> element contains the following elements:

Table 64. Elements contained within the <stop> element

Element	Required or optional?
<dateTime>	1 of these elements is required, and only 1 occurrence of the chosen element is allowed.
<never>	
<after>	

stopAfter element

The <stopAfter> element specifies the duration of time that a rule instance, as defined by the <groupingKey> element, is to remain active once it becomes active. After this duration of time, the rule instance is to be deactivated.

Attributes

The <stopAfter> element has the following attributes:

Table 65. Attributes of the <stopAfter> element

Name	Description	Data type	Required?
duration	Specifies the amount of time for the duration. The data type of this attribute is dependent on the value of the unit attribute.	• If the value of the unit attribute is `ISO-8601`, the data type is xsd:duration. • If the value of the unit attribute is `milliseconds`, the data type is xsd:positiveInteger.	Yes
unit	Specifies the unit of time to use. The valid values for this attribute are: • `ISO-8601` • `milliseconds`	xsd:string	Yes

The use of the ISO 8601 standard for time duration

Coding `ISO-8601` as the value for the unit attribute indicates that the value of the duration attribute is coded according to the ISO 8601 standard for specifying a time duration as one string. The standard XML schema data type specification uses ISO 8601 to provide a data type called `duration`. This data type is described in detail at http://www.w3.org/TR/xmlschema-2/#duration.

The format for the `duration` data type in the standard XML schema is the following string:

`PnYnMnDTnHnMnS`

- `P` is the character that always begins the string.
- *n*`Y` represents the number of years. A year is the same as 365 days. Therefore, coding `1Y` is the same as coding `365D`.
- *n*`M` represents the number of months. A month is the same as 30 days. Therefore, coding `1M` is the same as coding `30D`.

- *n*D represents the number of days.
- T is a separator that separates day units (years, months, and days) from time units (hours, minutes, and seconds). Time units always follow T.
- *n*H represents the number of hours.
- *n*M represents the number of minutes.
- *n*S represents the number of seconds.

Following are examples of the format:

- P5DT12H is 5.5 days.
- PT59M59S is 59 minutes and 59 seconds.
- P1M is 1 month.

Contained within

<stopAfter> is contained within the <activateOnEvent> element but only when <activateOnEvent> is coded within the <activationByGroupingKey> element.

Contains

<stopAfter> contains no elements.

thresholdRule element

The <thresholdRule> element defines a rule according to the threshold pattern.

Attributes

<thresholdRule> has the following attributes:

Table 66. Attributes of the <thresholdRule> element

Name	Description	Data type	Required?
name	Identifies the rule. This identifier must be unique within the rule block that contains this rule. It cannot contain a period.	xsd:NMTOKEN	Yes
processOnlyForwardedEvents	Determines whether the rule receives all events or only events that are forwarded from other rules. The default value is false, which indicates that the rule receives all events, including those that are forwarded from other rules.	xsd:boolean	No

Contained within

<thresholdRule> is contained within the following element:

- <ruleBlock>

Contains

<thresholdRule> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 67. Elements contained within the <thresholdRule> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<activationInterval>	Optional. 0 or 1 occurrence is allowed.
<lifeCycleActions>	Optional. 0 or 1 occurrence is allowed.
<eventSelector>	Optional. 0 or 1 occurrence is allowed.
<groupingKey>	Optional. 0 or 1 occurrence is allowed.
<booleanThreshold>	1 of these elements is required, and only 1 occurrence of the chosen element is allowed.
<computedThreshold>	
<eventCountThreshold>	
<timeWindow>	Required. Only 1 occurrence is allowed.
<onDetection>	Optional. 0 or 1 occurrence is allowed.
<onTimeOut>	Optional. 0 or 1 occurrence is allowed.

Related concepts

"Threshold pattern" on page 16
A threshold rule is defined by the threshold pattern. It collects a group of selected events within a time interval and determines, after each event is received, whether a threshold condition has been met. It is a stateful rule.

timeInterval element

The <timeInterval> element specifies the duration for the time window.

Attributes

<timeInterval> has the following attributes:

Table 68. Attributes of the <timeInterval> element

Name	Description	Data type	Required?
duration	Specifies the amount of time for the duration. The data type of this attribute is dependent on the value of the unit attribute.	• If the value of the unit attribute is `ISO-8601`, the data type is xsd:duration. • If the value of the unit attribute is `milliseconds`, the data type is xsd:positiveInteger.	Yes

Table 68. Attributes of the <timeInterval> element (continued)

Name	Description	Data type	Required?
unit	Specifies the unit of time to use. The valid values for this attribute are: • `ISO-8601` • `milliseconds`	xsd:string	Yes

The use of the ISO 8601 standard for time duration

Coding `ISO-8601` as the value for the unit attribute indicates that the value of the duration attribute is coded according to the ISO 8601 standard for specifying a time duration as one string. The standard XML schema data type specification uses ISO 8601 to provide a data type called `duration`. This data type is described in detail at http://www.w3.org/TR/xmlschema-2/#duration.

The format for the `duration` data type in the standard XML schema is the following string:

`PnYnMnDTnHnMnS`

- `P` is the character that always begins the string.
- *n*`Y` represents the number of years. A year is the same as 365 days. Therefore, coding `1Y` is the same as coding `365D`.
- *n*`M` represents the number of months. A month is the same as 30 days. Therefore, coding `1M` is the same as coding `30D`.
- *n*`D` represents the number of days.
- `T` is a separator that separates day units (years, months, and days) from time units (hours, minutes, and seconds). Time units always follow `T`.
- *n*`H` represents the number of hours.
- *n*`M` represents the number of minutes.
- *n*`S` represents the number of seconds.

Following are examples of the format:

- `P5DT12H` is 5.5 days.
- `PT59M59S` is 59 minutes and 59 seconds.
- `P1M` is 1 month.

Contained within

<timeInterval> is contained within the following element:

- <timeWindow>

Contains

<timeInterval> contains no elements.

timerRule element

The <timerRule> element defines a rule according to the timer pattern.

Attributes

<timerRule> has the following attributes:

Table 69. Attributes of the <timerRule> element

Name	Description	Data type	Required?
name	Identifies the rule. This identifier must be unique within the rule block that contains this rule. It cannot contain a period.	xsd:NMTOKEN	Yes
processOnlyForwardedEvents	This attribute is ignored because the timer rule does not process events.	xsd:boolean	No
repeat	Defines whether the timer rule runs repeatedly until it is deactivated. The valid values are: • `true` (the default value) • `false` If the value is set to `false`, the rule runs through its time interval only once, executes the rule response action when the respective time window is complete, and stops. If the <timeWindow> element for the timer rule contains the <runUntilDeactivated> element, the repeat attribute is ignored.	xsd:boolean	No

Contained within

<timerRule> is contained within the following element:

- <ruleBlock>

Contains

<timerRule> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 70. Elements contained within the <timerRule> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<variable>	Optional. 0 or more occurrences are allowed.
<activationInterval>	Optional. 0 or 1 occurrence is allowed.
<lifeCycleActions>	Optional. 0 or 1 occurrence is allowed.
<timeWindow>	Required. Only 1 occurrence is allowed.

Table 70. Elements contained within the <timerRule> element (continued)

Element	Required or optional?
<onTimeWindowComplete>	Optional. 0 or 1 occurrence is allowed.

Related concepts

"Timer pattern" on page 18
A timer rule is defined by the timer pattern. It initiates actions at regular intervals. It is a stateful rule. Although a timer rule does not process events, it can be activated or deactivated by an event.

timeWindow element

The <timeWindow> element contains elements that define the time interval during which the rule is processing.

Details

For example, the time window for a duplicate rule defines how long the rule must check for events that are duplicates of the first event that is accepted. If the time window is 30 seconds, the duplicate rule processes all duplicate events that occur within 30 seconds of the first event that it accepts.

Attributes

<timeWindow> has no attributes.

Contained within

<timeWindow> is contained within the following elements:

- <collectionRule>
- <computationRule>
- <duplicateRule>
- <sequenceRule>
- <thresholdRule>
- <timerRule>

Contains

<timeWindow> contains the following elements:

Table 71. Elements contained within the <timeWindow> element

Element	Required or optional?
<timeInterval>	1 of these elements is required, and only 1 occurrence of the chosen element is allowed.
<runUntilDeactivated>	

variable element

The <variable> element defines a variable and contains information in a form that can be referenced by expressions. A variable can be defined at the level of the rule set, a rule block, or a rule.

Details

rule set variable
Applies globally to the rule set and can be referenced by any expression in that rule set.

rule block variable
Applies only within the rule block (and within any contained rule blocks) and can be referenced by any expression within that rule block.

rule variable
Applies only to expressions within that rule.

Variables can have the same name at different levels in the rule hierarchy. When a variable is accessed, the most local definition of the variable is used. For example, if a variable is defined at the rule set level, rule block level, and rule level with the same name, the variable definition at the rule level is used by the expressions within that rule.

When variables are defined at the rule set or rule block level, multiple rules get and set these variables at different times. Therefore, to ensure that variable values are maintained correctly, be aware of how you code the interactions among variables in the rule set.

If the variable is defined at the rule set or rule block level, it is not reinitialized after the rule pattern is matched.

Under either of the following conditions, use locking on the getting and setting of rule set and rule block variables to prevent the variable values from being incorrectly set:

- If the timer rule gets or sets a variable during an <onTimeOut> action
- If the application in which the Active Correlation Technology engine is embedded is multithreaded

If a rule is defined with a grouping key, rule variables that are defined by the <variable> element are not valid within life cycle actions or within a <filteringPredicate> element that is contained within an <activateOnEvent> or <deactivateOnEvent> element that is contained within an <activationInterval> element. This is because in this case, the rule variables apply only to a rule instance, and rule instances do not exist at the time that these expressions are run.

Attributes

<variable> has the following attributes:

Table 72. Attributes of the <variable> element

Name	Description	Data type	Required?
name	Identifies a specific variable. A variable is referenced by its name.	xsd:NMTOKEN	Yes
dataType	Identifies the type of information that a variable contains. This must be a fully qualified data type such as java.lang.String.	xsd:NMTOKEN	Yes

Name restrictions for variables

Variable names have certain restrictions. Therefore, the value of the name attribute on the <variable> element has the following restrictions:

- It can include only the following characters:
  - Uppercase ASCII Latin letters A-Z. The Unicode representation is `\u0041-\u005a`.
  - Lowercase ASCII Latin letters a-z. The Unicode representation is `\u0061-\u007a`.
  - The ASCII underscore (_). The Unicode representation is `\u005f`.
  - The dollar sign ($). The Unicode representation is `\u0024`.
  - The ASCII digits 0 – 9. The Unicode representation is `\u0030-\u0039`.
- It cannot be null.
- It cannot be an empty string.
- It cannot contain any blank spaces.
- It cannot contain a period.
- It cannot start with `act_` in any form (not in uppercase, lowercase, or mixed case).

Contained within

<variable> is contained within the following elements:

- <ruleSet>
- <ruleBlock>
- <collectionRule>
- <computationRule>
- <duplicateRule>
- <filterRule>
- <sequenceRule>
- <thresholdRule>
- <timerRule>

Contains

<variable> contains the following elements.

The elements must be coded in the order that is shown. If an element is optional, it does not need to be coded, but all elements that are coded must follow the correct order.

Table 73. Elements contained within the <variable> element

Element	Required or optional?
<comment>	Optional. 0 or 1 occurrence is allowed.
<varInitializer>	Required. 1 occurrence is allowed.

Related concepts

"Variables" on page 23
In the rule language, certain variables are used to store event-related information across different event occurrences or rules. This event-related information can then be accessed from expressions within rules. Some types of

variables are defined by the rule writer, and others are provided by the Active Correlation Technology. Some types can be accessed directly within an expression, and others can be accessed only through methods that are provided by the Active Correlation Technology.

varInitializer element

The <varInitializer> element contains an expression that provides the initial value for the variable that is defined in the associated <variable> element.

Details

Because the variable can be of any type, the expression code can return an array object or any other complex, implementation-specific object to be stored by the Active Correlation Technology engine.

Refer to "Variables" on page 23 for information about the variables that can be used in expressions. The use of certain variables is dependent on the context of the expression.

Attributes

<varInitializer> has the following attribute:

Table 74. Attributes of the <varInitializer> element

Name	Description	Data type	Required?
expressionLanguage	Identifies the programming language in which the expression is written. Because the Java programming language is the only supported expression language, the only valid value for this attribute is `java`.	xsd:NMTOKEN	Yes

Contained within

<varInitializer> is contained within the following element:

- <variable>

Contains

<varInitializer> contains no elements.

Related concepts

"Expressions" on page 19
An expression is code that contains custom logic that can be added to a rule. Expressions can also access code that is external to the Active Correlation Technology engine. In the rule language, expressions are valid only within specific contexts, or rule language elements.

whenLoaded element

The <whenLoaded> element specifies that a rule is activated when it is loaded by the Active Correlation Technology engine.

Attributes

<whenLoaded> has no attributes.

Contained within

<whenLoaded> is contained within the following element:

- <start>

Contains

<whenLoaded> contains no elements.

Chapter 6. Glossary

This glossary contains the terms and definitions for important concepts in the Active Correlation Technology.

ACT See Active Correlation Technology.

action An expression that runs as part of a rule response or when a rule is loaded, unloaded, activated, or deactivated.

Active Correlation Technology
An IBM technology that provides event correlation through rules.

Active Correlation Technology compiler
The Active Correlation Technology component that parses a rule set and any code that is contained within it to generate the internal data structures that are required by the Active Correlation Technology engine.

Active Correlation Technology engine
The Active Correlation Technology component that processes events according to the output of the Active Correlation Technology compiler.

Active Correlation Technology rule builder
A GUI for writing correlation rules in the Active Correlation Technology rule language.

Active Correlation Technology rule language
An XML-based language for writing rules to correlate events. These rules can then be deployed to Active Correlation Technology runtime environments.

Active Correlation Technology runtime environment
An application in which the Active Correlation Technology engine is embedded, with or without the compiler.

collection pattern
A rule pattern that defines a rule to collect a group of selected events within a time interval. A rule that is defined by the collection pattern is a stateful rule.

computation pattern
A rule pattern that defines a rule to apply a computation (through an expression) to collected events as each event is received within a time interval. A rule that is defined by the computation pattern is a stateful rule.

domain
The category to which a group of rules apply based on their function. For example, a domain can represent a specific geographic area, IT management discipline (such as security detection or network event correlation), or business organization (such as a specific company or division with a company).

duplicate pattern
A rule pattern that defines a rule to count the second and subsequent events that are accepted within the specified time interval but to skip the rule set processing for these events. A rule that is defined by the duplicate pattern is a stateful rule.

event provider
Any software that generates events that are processed by the Active Correlation Technology.

event selector
The criteria for event selection. These criteria determine what events are accepted for processing by a rule. The event selector includes the event type and the filtering predicate.

expression
Code that contains custom logic that can be added to a rule. Rule writers can use expressions for different purposes, such as the initialization of variables, the definition of event selection criteria, or the specification of rule response actions and life cycle actions.

expression language
The programming language in which an expression is written.

external event
An event that the Active Correlation Technology engine receives from a source that is external to it.

external object
An object that an application creates to communicate with expressions.

filtering predicate
An expression that defines the condition under which an event is accepted for processing by a rule. The filtering predicate is a part of an event selector. A filtering predicate returns a boolean value.

filter pattern
A rule pattern that defines a rule to take a certain action when it accepts an event. A rule that is defined by the filter pattern acts only on a single event and is therefore a stateless rule.

grouping key
A method for directing a rule to create a separate rule instance (or copy of itself) for each group of events that share common characteristics.

import
A programming language-specific way to make external code accessible to expressions.

internal event
An event that is created by a rule that is running in the Active Correlation Technology engine. This event might be forwarded to other rules.

life cycle action
An expression that runs when a rule is loaded, unloaded, activated, or deactivated.

node An object within the rule hierarchy that can be individually and independently added to, removed from, or replaced within a rule set. Specifically, these objects are nodes:

- Rules
- Rule blocks
- Rule block variables
- Rule set variables

Because an object cannot be operated on individually and independently below the rule level, a rule variable is not a node.

predicate
See filtering predicate.

response
See rule response.

rule The correlation unit that is used to recognize relationships among events and to run appropriate rule responses. A rule is an implementation of one of seven rule patterns and is organized, according to its function, into a rule block that is part of a rule set. A rule accepts an event for processing if the event meets the event selection criteria.

rule block
The organizational unit for grouping rules by function into domains within the rule set. A rule block can contain not only rules but also other rule blocks.

rule instance
In the context of the grouping key, a copy of a rule.

rule pattern
The representation of an event correlation situation (such as a threshold condition or duplicate event detection). The Active Correlation Technology rule language includes the following rule patterns: collection, computation, duplicate, filter, sequence, threshold, and timer. The pattern of a rule is matched when the situation that is defined by the rule occurs. When the pattern is matched, the rule concludes its processing by taking the appropriate rule response actions. While a rule is active, the rule pattern can be matched multiple times.

rule response
An expression that is run when the Active Correlation Technology engine recognizes that a rule condition has been met. A rule response consists of one or more actions.

rule response action
See action.

rule set
The rule execution unit for the Active Correlation Technology rule language. The rule set contains the rules, organized into rule blocks, to be executed by the Active Correlation Technology engine. The engine acts on only one rule set at a given time.

sequence pattern
A rule pattern that defines a rule to detect the presence or absence of a certain sequence of events within a time interval. The sequence can be ordered or random. A rule that is defined by the sequence pattern is a stateful rule.

snippet
An excerpt of source code.

stateful rule
A rule that retains state information, which is information about the characteristics of a rule instance, for the purpose of acting on a collection of events over a period of time. Rules that are defined by any of the following rule patterns are stateful rules: collection, computation, duplicate, sequence, threshold, or timer.

stateless rule
A rule that does not retain state information and therefore can act only on one event at a time. A rule that is defined by the filter pattern is a stateless rule.

threshold pattern
A rule pattern that defines a rule to collect a group of selected events within a time interval and to determine, after each event is received, whether a threshold condition has been met. A rule that is defined by the threshold pattern is a stateful rule.

timer pattern
A rule pattern that defines a rule to initiate actions at regular intervals. A rule that is defined by the timer pattern is a stateful rule. Although a timer rule does not process events, it can be activated or deactivated by an event.

Appendix. Notices

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Trademarks

DB2, IBM, the IBM logo, Tivoli, the Tivoli logo, Tivoli Enterprise Console, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Printed in USA