Event processing

This section describes configuration of event routing (Groovy rules).

The `eventRule` method helps to build a rule for event processing. It requires a few components for its instantiation:

```
eventRule(String description, Class<AppEvent> eventType, Predicate<AppEvent> ruleCondition, Consumer<AppEvent> ruleAction)
```

- **description** - string with free text description of the rule
- **event type** - type or subtype of the event for which the rule is applied
- **rule condition** - dynamic filter, which at the same time can check appliance of this rule depending on the event content and source attributes. The context filter can be NULL if you'd like to ignore it.
- **rule action** - describes the main goal of the rule

EventType

The event type is a type of the events for which the rule is applied. Subtype or general type can be specified.

The table below specifies the available list of types and their methods.

Event category	Class of event	Methods	Description
General			
	`BasicAppEvent`		The basic class of all events that can be used for handling all the events the system throws.
Application			
	`ServerStateEvent`	getSource(): STARTED	Notifies about changes in the state of the FIX server.
Session			
	`FIXSessionStateEvent`	getSessionState(): CONNECTING, WAITING_FOR_LOGON, CONNECTED, WAITING_FOR_LOGOFF, DISCONNECTED, LOGON_RECEIVED, DEAD, DISCONNECTED_ABNORMALLY, RECONNECTING, WAITING_FOR_FORCED_LOGOFF, WAITING_FOR_FORCED_DISCONNECT getSessionId(): String	Notifies about changes in the state of FIX sessions.
	`NewSessionEvent`	SessionParameters getSessionParameters() reject(String reason) reject(String reason, boolean quiet)	The event is intended for accepting or rejecting a new incoming session. The session will be rejected if the reject method is called. Otherwise, the session will be accepted. **NOTE**: If the reject method is called, the next rule for the event will not be processed.
Scheduler			
	`SchedulerEvent`	getId(): String	The event that is created by the Scheduler when the trigger executes.
SnF			
	SnFEvent	getSource(): MESSAGE_WAS_QUEUED, MESSAGE_WAS_SKIPPED	
Rule events			
	`RuleErrorEvent`	getException(): Exception getMessage(): FIXFieldList getRuleDescription(): String	Notifies that there is an exception in some of the rules processed.
	`UnprocessedMessageEvent`	getMessage(): FIXFieldList	Notifies that there are no acceptable rules defined for the specified message.

NOTE: If `BasicAppEvent` is specified in the **eventRule** method as an event type parameter, all the events above will trigger the rule.

Examples

```
import com.epam.fej.server.fix.event.ServerStateEvent
import com.epam.fej.server.fix.event.SessionStateEvent
import com.epam.fej.scheduling.event.SchedulerEvent

[
        eventRule("Catching session events", SessionStateEvent.class, {
                appEvent ->
                    return true//do nothing but there can be additional logic
            },
            {
                sessionStateEvent ->
                    def sessionName = sessionStateEvent.getSessionId()
                    logger.info("I'm rule for session events, I was called because session '{}' has been
started.", sessionName)
            }
        ),
        eventRule("Catching server events", ServerStateEvent.class, {
                appEvent -> return true//do nothing but there can be additional logic
            },
            {
                serverStateEvent ->
                    String state = String.valueOf(serverStateEvent.getSource())
                    logger.info("I'm rule for server events, I was called because server has {}.", state)
            }
        ),
        eventRule("Catching scheduler events", SchedulerEvent.class, {
                appEvent -> return true//do nothing but there can be additional logic
            },
            {
                schedulerEvent ->
                    logger.info("I'm rule for schedule events, id - {}.", schedulerEvent.getId())
            }
        ),
                eventRule("Custom accepting of FIX sessions", NewSessionEvent.class, null,
                {
                    newSessionEvent ->
                        boolean accepted = true
                        def sessionParameters = newSessionEvent.getSessionParameters()
                        if (sessionParameters.getIncomingUserName() != "User1" || sessionParameters.
getIncomingPassword() != "Password1") {
                            newSessionEvent.reject("Wrong username or/and password")
                            accepted = false
                        }
                        logger.info("Session {} is {}", sessionParameters.getSessionID(), accepted ? "accepted"
: "declined")
                }
        )
]
```

The example of the Scheduler settings: the **SchedulerEvent** event will be published every 10 seconds.

```
<schedule id="every10Seconds">
        <task name="event" timeZone="Europe/Samara">
            <event cron="*/10 * * ? * *"/>
        </task>
</schedule>
```