User Guide

Table of Contents

Motivation

Motivation

Typed access to process variables

Camunda BPM engine provide Java API to access the process variables. This consists of:

- `RuntimeService` methods
- `TaskService` methods
- Methods on `DelegateExecution`
- Methods on `DelegateTask`
- `VariableMap`

All those methods requires the user of the API to know the variable type. Here is a usage example:

```
ProcessInstance processInstance = ...;
List<OrderPosition> orderPositions = (List<OrderPosition>) runtimeService
  .getVariable(processInstance.id, "orderPositions");
```

This leads to problems during refactoring and makes variable access more complicated than it is. This library addresses this issue and allows for more convenient type-safe process variable access.

More details can be found in:

- Data in Process (Part 1)
- Data in Process (Part 2)

Variable guards

Process automation often follows strict rules defined by the business. On the other hand, the process execution itself defines rules in terms of pre- and post-conditions on the process payload (stored as process variables in Camunda BPM). Rising complexity of the implemented processes makes the compliance to those rules challenging. In order to fulfill the conditions on process variables during the execution of business processes, a concept of `VariableGuard` is provided by the library. A guard consists of a set of `VariableConditions` and can be evaluated in all contexts, the variables are used in: `DelegateTask`, `DelegateExecution`, `TaskService`, `RuntimeService`, `VariableMap`.

Here is an example of a task listener verifying that a process variable `ORDER_APPROVED` is set, which will throw a `GuardViolationException` if the condition is not met.

```
import static io.holunda.camunda.bpm.data.guard.CamundaBpmDataGuards.exists;

@Component
class MyGuardListener extends AbstractGuardTaskListener {

    public MyGuardListener() {
        super(newArrayList(exists(ORDER_APPROVED)), true);
    }
}
```

Features

Features

- Process Variables
    - The library provides a way to construct generic adapter for every process variable.
    - The adapter contains variable name.
    - The adapter contains variable type.
    - The adapter can be applied in any context (`RuntimeService`, `TaskService`, `DelegateExecution`, `DelegateTask`, `VariableMap`).
    - The adapter offers methods to read, write, update and remove variable values.
    - The adapter works for all types supported by Camunda BPM. This includes primitive types, object and container types (`List<T>`, `Set<T>`, `Map<K , V>`).
    - The adapter supports global / local variables.
    - The adapter support transient variables.
    - Fluent builders are available in order to set, remove or update multiple variables in the same context.
- Process Variable Guards
    - Generic support for `VariableGuard` for evaluation of a list of `VariableCondition`s
    - Condition to check if variable exists.
    - Condition to check if variable doesn’t exist
    - Condition to check if variable has a predefined value.
    - Condition to check if variable has one of predefined values.
    - Condition to check if variable matches condition specified by a custom function.
    - `AbstractGuardTaskListener` to construct variable conditions guards easily.
    - `AbstractGuardExecutionListener` to construct variable conditions guards easily.
- Testing variable access and mocking `RuntimeService` and `TaskService`.
    - Builder to create Mockito-based behaviour of `RuntimeService` accessing variables.
    - Builder to create Mockito-based behaviour of `TaskServiceService` accessing variables.

Java Examples

Java Examples

The following example code demonstrates the usage of the library using Java.

Define variable

```
public class OrderApproval {
  public static final VariableFactory<String> ORDER_ID = stringVariable("orderId");
  public static final VariableFactory<Order> ORDER = customVariable("order", Order.class
  public static final VariableFactory<Boolean> ORDER_APPROVED = booleanVariable("orderApp
  public static final VariableFactory<OrderPosition> ORDER_POSITION = customVariable("or
  public static final VariableFactory<BigDecimal> ORDER_TOTAL = customVariable("orderTot
}
```

Read variable from Java delegate

```
@Configuration
class JavaDelegates {

  @Bean
  public JavaDelegate calculateOrderPositions() {
    return execution -> {
      OrderPosition orderPosition = ORDER_POSITION.from(execution).get();
      Boolean orderApproved = ORDER_APPROVED.from(execution).getLocal();
      Optional<BigDecimal> orderTotal = ORDER_TOTAL.from(execution).getOptional();
    };
  }
}
```

Write variable from Java delegate

```
import java.math.BigDecimal
;@Configuration
class JavaDelegates {

  @Bean
  public JavaDelegate calculateOrderPositions() {
    return execution -> {
      OrderPosition orderPosition = new OrderPosition("Pencil", BigDecimal.valueOf(1.5),
      ORDER_POSITION.on(execution).set(orderPosition);
    };
  }
}
```

Remove variable from Java delegate

```
import java.math.BigDecimal
;@Configuration
class JavaDelegates {

  @Bean
  public JavaDelegate calculateOrderPositions() {
    return execution -> {
      ORDER_APPROVED.on(execution).removeLocal();
    };
  }
}
```

Update variable from Java delegate

```
import java.math.BigDecimal;
@Configuration
class JavaDelegates {

  @Bean
  public JavaDelegate calculateOrderPositions() {
    return execution -> {
      OrderPosition orderPosition = ORDER_POSITION.from(execution).get();
      ORDER_TOTAL.on(execution).updateLocal(amount -> amount.add(orderPosition.getNetCos
    };
  }
}
```

Fluent API to remove several variables

```
@Configuration
class JavaDelegates {

  @Bean
  public ExecutionListener removeProcessVariables() {
    return execution ->
    {
      CamundaBpmData.builder(execution)
          .remove(ORDER_ID)
          .remove(ORDER)
          .remove(ORDER_APPROVED)
          .remove(ORDER_TOTAL)
          .removeLocal(ORDER_POSITIONS);
    };
  }
}
```

Fluent API to set several variables

```
@Component
class SomeService {

  @Autowired
  private RuntimeService runtimeService;
  @Autowired
  private TaskService taskService;

  public void setNewValuesForExecution(String executionId, String orderId, Boolean order
      CamundaBpmData.builder(runtimeService, executionId)
          .set(ORDER_ID, orderId)
          .set(ORDER_APPROVED, orderApproved)
          .update(ORDER_TOTAL, amount -> amount.add(10));
  }

  public void setNewValuesForTask(String taskId, String orderId, Boolean orderApproved)
      CamundaBpmData.builder(taskService, taskId)
          .set(ORDER_ID, orderId)
          .set(ORDER_APPROVED, orderApproved);
  }

  public void start() {
      VariableMap variables = CamundaBpmData.builder()
          .set(ORDER_ID, "4711")
          .set(ORDER_APPROVED, false)
          .build();
      runtimeService.startProcessInstanceById("myId", "businessKey", variables);
  }
}
```

Example project

For more examples, please check-out the Java Example project, at Github

Kotlin Examples

Kotlin Examples

The following snippets demonstrate the usage of the library from Kotlin

Define variable

```
import io.holunda.data.CamundaBpmDataKotlin

object Variables {
    val ORDER_ID = stringVariable("orderId")
    val ORDER: VariableFactory<Order> = customVariable("order")
    val ORDER_APPROVED = booleanVariable("orderApproved")
    val ORDER_POSITION: VariableFactory<OrderPosition> = customVariable("orderPosition")
    val ORDER_TOTAL: VariableFactory<BigDecimal> = customVariable("orderTotal")
}
```

Read variable from Java delegate

```
@Configuration
class JavaDelegates {

    @Bean
    fun calculateOrderPositions() = JavaDelegate { execution ->
        val orderPosition = ORDER_POSITION.from(execution).get()
        // order position is of type OrderPosition
    }
}
```

Write variable from Java delegate

```
import java.math.BigDecimal

@Configuration
class JavaDelegates {

    @Bean
    fun calculateOrderPositions() = JavaDelegate { execution ->
        val orderPosition = ORDER_POSITION.from(execution).get()
        ORDER_TOTAL.on(execution).set {
            orderPosition.netCost.times(BigDecimal.valueOf(orderPosition.amount))
        }
    }
}
```

Remove variable from Java delegate

```
@Configuration
class JavaDelegates {

    @Bean
    fun removeTotal() = JavaDelegate { execution ->
        ORDER_TOTAL.on(execution).remove()
    }
}
```

Update variable from Java delegate

```
import java.math.BigDecimal
@Configuration
```

```
class JavaDelegates {

    @Bean
    fun calculateOrderPositions() = JavaDelegate { execution ->
        val orderPosition = ORDER_POSITION.from(execution).get()
        ORDER_TOTAL.on(execution).update {
            it.plus(orderPosition.netCost.times(BigDecimal.valueOf(orderPosition.amount))
        }
    }
}
```

Fluent API to remove several variables

```
import io.holunda.camunda.bpm.data.remove

@Configuration
class JavaDelegates {

    @Bean
    fun removeProcessVariables() = JavaDelegate { execution ->
        execution
            .remove(ORDER_ID)
            .remove(ORDER)
            .remove(ORDER_APPROVED)
            .remove(ORDER_TOTAL)
            .removeLocal(ORDER_POSITIONS)
    }
}
```

Fluent API to set several variables

```
@Component
class SomeService(
    private val runtimeService: RuntimeService,
    private val taskService: TaskService
) {

    fun setNewValuesForExecution(executionId: String, rderId: String, orderApproved: Boo
        runtimeService.builder(executionId)
            .set(ORDER_ID, orderId)
            .set(ORDER_APPROVED, orderApproved)
            .update(ORDER_TOTAL, { amount -> amount.add(10) })
    }

    fun setNewValuesForTask(taskId: String, orderId: String, orderApproved: Boolean) {
        taskService.builder(taskId)
            .set(ORDER_ID, orderId)
            .set(ORDER_APPROVED, orderApproved)
    }

  fun start() {
      val variables = createProcessVariables()
          .set(ORDER_ID, "4711")
          .set(ORDER_APPROVED, false)
      runtimeService.startProcessInstanceById("myId", "businessKey", variables)
  }
}
```

Example project

For more examples, please check-out the Kotlin Example project, at Github.

Further outlook

Further outlook

- Implement Contracts to be able to check a guards automatically
- Implement an Anti-Corruption-Layer to protect Message correlation and External Task completion from setting wrong values (unguarded).