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:

```
List<OrderPosition> orderPositions = (List<OrderPosition>) runtimeService
  .getVariable("myExecutionIn", "orderPositions");
```

This leads to problems during refactoring and makes variable access more complicated than it is. More details can be found in:

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

This library addresses this issue and allows for more convenient type-safe process variable access.

Features

Features

The library Camunda BPM Data provides the following functionality:

- 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.

Examples

Examples

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("orc
  public static final VariableFactory<BigDecimal> ORDER_TOTAL = customVariable("orderTota
}
```

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.getNetCost
    };
  }
}
```

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 orderA
      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 projects

We provide two examples:

- Java Example, see Github
- Kotlin Example, see Github

Further outlook

Further outlook

Planned or featured features:

- Native Kotlin support including extension functions
- Process variable guards for better testing