d.velop

# d.velop process (Cloud): Administrieren

# Inhaltsverzeichnis

1. Administrationshandbuch d.velop process (Cloud) ..... 3
1.1. Basisinformationen zur Anwendung und zum Handbuch ..... 3
1.1.1. Über d.velop process ..... 3
1.2. Konfigurieren von d.velop process ..... 3
1.2.1. Einrichten von Benutzerrollen (optional) ..... 3
1.2.2. Konfigurieren Protokollexporten ..... 3
1.3. Administrieren von Prozessen ..... 4
1.3.1. Wissenswertes zum Feature "Prozessadministration" ..... 4
1.3.2. Anzeigen aller Prozessinstanzen ..... 5
1.3.3. Anzeigen eines Instanzdiagramms ..... 5
1.3.4. Ändern der Variablen eines Tokens ..... 5
1.3.5. Erneutes Ausführen eines fehlgeschlagenen Tokens ..... 5
1.3.6. Erneutes Ausführen eines asynchronen Services ..... 6
1.3.7. Wissenswertes zur Protokollierung von Prozessen ..... 6
1.3.8. Aktivieren der Protokollierung für eine Prozessdefinition ..... 7
1.3.9. Erstellen und konfigurieren eines Prozessereignisses ..... 7
1.3.10. Wissenswertes zu Ausdrücken in Prozessereignissen ..... 8
1.4. Modellieren von Prozessen ..... 10
1.4.1. Wissenswertes zur Prozessmodellierung ..... 10
1.4.2. Verwenden von BPMN-Elementen ..... 10
1.4.3. Erstellen eines exemplarischen Prozesses in einer BPMN-Datei ..... 11
1.4.4. Verwenden von Ausdrücken ..... 14
1.4.5. Arbeiten mit Prozessvariablen ..... 15
1.4.6. Verwenden von Timer-Ereignissen ..... 21
1.4.7. Verwenden von Verzweigungen ..... 25
1.4.8. Einfügen von Benutzeraktivitäten ..... 29
1.4.9. Verwenden einer Multi-Instanz ..... 38
1.4.10. Verwenden eines Startformulars ..... 40
1.4.11. Verwenden von Services ..... 40
1.4.12. Verwenden von Eskalationen ..... 51
1.4.13. Verändern von Prozessvariablen einer Multi-Instanz ..... 57
1.4.14. Verändern von Prozessvariablen ..... 60
1.5. Häufig gestellte Fragen ..... 61
1.5.1. Warum tritt beim Einfügen einer Variablen ein unbekannter Serverfehler auf? .. 61
1.5.2. Wo finde ich eine Übersicht über alle Aktionen? ..... 61
1.6. Weitere Informationsquellen und Impressum ..... 61

# 1. Administrationshandbuch d.velop process (Cloud)

## 1.1. Basisinformationen zur Anwendung und zum Handbuch

In diesem Kapitel finden Sie Produkthinweise und allgemeine Informationen.

### 1.1.1. Über d.velop process

d.velop process ist eine Anwendung, mit der Sie Prozesse ausführen, überwachen und administrieren können.

Mit d.velop process sind Sie in der Lage, Geschäftsprozesse zu automatisieren. Sie können Anwender in Form von Aufgaben an Prozessen teilnehmen lassen oder automatisierte Serviceaktivitäten ausführen. d.velop process unterstützt den BPMN-Standard. Dadurch haben Sie vielfältige Möglichkeiten bei der Gestaltung von Prozessen.

In der Prozessüberwachung haben Sie die Möglichkeit, den Status aktiver Prozesse zu kontrollieren. Im Falle eines Fehlers haben Sie direkt die Möglichkeit, Korrekturen vorzunehmen. Damit stellen Sie auf einfache Weise den erfolgreichen Betrieb Ihrer Geschäftsprozesse sicher.

Die Prozessadministration unterstützt Sie dabei, Aktionen über mehrere Prozessinstanzen durchzuführen. Sie können beispielsweise mehrere Prozessinstanzen abbrechen, diese bei Prozessänderungen in eine neue Version migrieren, fehlerhafte Prozessinstanzen erneut ausführen oder eine Prozessversion löschen.

Wenn Sie einen eigenen Prozess modelliert haben, können Sie diesen in der Prozessadministration bereitstellen. Der Prozess steht Ihnen dann in der d.3ecm-Welt zur Verfügung.

## 1.2. Konfigurieren von d.velop process

In diesem Thema finden Sie Informationen zur Konfiguration und weiteren Einstellungen.

### 1.2.1. Einrichten von Benutzerrollen (optional)

Mit dem Eintrag **Benutzerrollen** im Feature **Konfiguration** können Sie Benutzern eine Benutzerrolle zuweisen. Jeder authentifizierte Benutzer ist nach der Einrichtung von d.velop process automatisch der Rolle Prozessbenutzer zugewiesen.

Sie können Benutzern folgende Rollen zuweisen:

- **Prozessadministrator**: Besitzt alle Berechtigungen für die Arbeit mit d.velop process.
- **Prozessbenutzer**: Darf Einmalprozesse starten und Prozesse bereitstellen.

Möchten Sie einem Benutzer beispielsweise die Rolle Prozessadministrator zuweisen, tragen Sie einfach den Benutzernamen in das entsprechende Feld ein und wählen ihn aus.

### 1.2.2. Konfigurieren Protokollexporten

Sie möchten den Export von Prozessprotokollen in ein d.3-Repository konfigurieren, um Protokolle langfristig zu speichern.

**Das müssen Sie wissen**

- Sie benötigen ein konfiguriertes d.3-Repository, in dem die Protokollexporte gespeichert werden.
- Sie müssen mit einem Benutzer angemeldet sein, der Zuordnungen für das d.3-Repository administrieren kann.

- Sie müssen in Ihrem Repository eine Dokumentart erstellen, die vier Eigenschaften vom Typ **A lphanumerisch** (je nach System auch **Text**) mit einer maximalen Länge von 250 Zeichen besitzt. Diese neuen Eigenschaften müssen Sie im Folgenden den Prozesseigenschaften zuordnen. Geben Sie den Eigenschaften sprechende Namen, die zu den Prozesseigenschaften passen, z.B. **Prozess**, **Prozessinstanz**, **Business-Key** und **Anhang**.
- Wenn einer der Werte länger als 250 Zeichen (oder 255 Bytes) ist, wird dieser Wert beim Export passend abgeschnitten. Die abgeschnittene Stelle wird mit drei Punkten gekennzeichnet.
- Wenn es beim Exportvorgang zu einer Kürzung von Metadaten oder einem Fehler kommt, sehen Sie einen entsprechenden Hinweis für die Prozessinstanz in der Prozessüberwachung sowie in der Protokollansicht.
- Sie müssen einen API-Schlüssel für einen Benutzer erstellen, der Administrationsrechte besitzt. Darüber hinaus benötigt der Benutzer lesenden und schreibenden Zugriff für diese Dokumentart.

**So geht's**

1. Öffnen Sie das Feature **Zuordnungen**.
2. Fügen Sie eine neue Zuordnung hinzu.
3. Wählen Sie die Quelle **Prozesse** aus.
4. Fügen Sie eine Kategorie hinzu und verknüpfen Sie die Quelle **Prozessprotokoll** mit der gewünschten Dokumentart.
5. Verknüpfen Sie alle Eigenschaften der Quelle mit Dokumenteigenschaften des Ziels. Stellen Sie sicher, dass alle vier Eigenschaften zugeordnet sind.
6. Speichern Sie die Zuordnung.
7. Öffnen Sie das Feature **Konfiguration** und wählen Sie **DMS-Exportkonfiguration** im Bereich **Prozesseinstellungen** aus.
8. Wählen Sie das Zielrepository aus und tragen Sie den API-Schlüssel ein.
9. Speichern Sie die Änderungen.

Nachdem Sie den Protokollexport konfiguriert haben, können Sie die Option zum Exportieren des Protokolls beim Bereitstellen einer Prozessdefinition aktivieren.

# 1.3. Administrieren von Prozessen

In diesem Thema erfahren Sie, wie Sie Ihre Prozesse verwalten können. Informieren Sie sich über die Möglichkeiten, einen Überblick über Ihre Prozesse zu erhalten und fehlerhafte Prozesse zu korrigieren.

## 1.3.1. Wissenswertes zum Feature "Prozessadministration"

Mit dem Feature **Prozessadministration** können Sie Aktionen starten, die ganze Prozessversionen betreffen. In der Regel umfassen die Aktionen mehrere Prozessinstanzen. Die Prozessadministration erlaubt es Ihnen, Aktionen zu verschiedenen Prozessen und Versionen nacheinander durchzuführen, ohne die Ansicht wechseln zu müssen.

Sie können folgende Aktionen für ganze Prozessversionen durchführen:

- **Prozessversionen erneut ausführen**: Wenn innerhalb eines Prozesses mehrere Fehler auftreten, können Sie zur Fehlerbehebung die Prozessversion erneut ausführen. Verwenden Sie zum erneuten Ausführen einer Prozessversion die Aktion **Erneut ausführen**.
- **Prozessinstanzen in die nächste Prozessversion migrieren**: Bei der Aktualisierung von Prozessen wird automatisch eine neue Version des Prozesses angelegt. Sie können mit der Aktion **Migrieren** alle Instanzen einer Prozessversion in die neuste Version migrieren.
- **Prozessversionen löschen**: Wenn keine laufenden Prozessinstanzen zu einer Version mehr vorliegen, können Sie die Prozessversion löschen. Verwenden Sie zum Löschen einer Prozessversion die Aktion **Prozessversion löschen**.
- **Prozessinstanzen abbrechen**: Sie können alle offenen Instanzen einer Prozessversion abbrechen. Durch das Abbrechen werden alle offenen Aktivitäten zu dieser Prozessversion entfernt und die Prozessinstanz kann nicht wieder fortgesetzt werden. Verwenden Sie zum Abbrechen einer Prozessinstanz die Aktion **Instanzen abbrechen**.

Darüber hinaus stehen noch folgende Aktionen zur Verfügung:

- **Neuen Prozess bereitstellen**: Sie können einen neuen Prozess (oder eine neue Version eines bereits bestehenden Prozesses) aus einer **.bpmn**-Datei ins System importieren.

### 1.3.2. Anzeigen aller Prozessinstanzen

Mit dem Feature **Prozessüberwachung** erhalten Sie einen Überblick über alle Prozessinstanzen. Sie können mit dem Business-Key gezielt nach einzelnen Prozessen suchen oder die Ergebnisliste mit den Filtern **Prozesse**, **Versionen**, **Aktivitäten** und **Status** eingrenzen. Ein Prozess befindet sich in einem der folgenden vier Status:

 : Die Instanz wurde abgebrochen.

 : Die Instanz ist in einem Fehlerzustand.

 : Die Instanz wurde erfolgreich beendet.

 : Die Instanz läuft aktuell noch.

Ein zusätzliches Ausrufungszeichen beim Status signalisiert einen Fehler oder eine Warnung beim Prozessprotokollexport.

Um nähere Informationen zu einer Prozessinstanz zu erhalten, wählen Sie den Prozess aus. In der Detailansicht stehen Ihnen Kontextaktionen zur Verfügung, die sich je nach Prozesstyp unterscheiden können. Mit den Kontextaktionen können Sie die Prozessinstanz abbrechen, in die nächste Version migrieren, in einem Instanzdiagramm darstellen oder in der Quelle anzeigen. Für eine Übersicht über den Prozessverlauf können Sie außerdem das Prozessprotokoll anzeigen.

### 1.3.3. Anzeigen eines Instanzdiagramms

In der Detailansicht einer Instanz können Sie die Prozessinstanz mit der Kontextaktion **Instanzdiagramm** in einem Diagramm anzeigen.

Um bei sehr komplexen Instanzdiagrammen den Überblick zu bewahren, können Sie die Ansicht des Diagramms ändern. Mit dem Mausrad können Sie die Größe des Diagramms ändern. Wenn Sie die Ansicht mit dem Mausrad vergrößert haben, können Sie das Diagramm mit gedrückter linker Maustaste verschieben. Mit **Ansicht zurücksetzen** kehren Sie zur ursprünglichen Ansicht zurück.

### 1.3.4. Ändern der Variablen eines Tokens

Wenn Sie im Feature **Prozessüberwachung** in die Detailansicht eines Tokens navigieren, können Sie die Variablen der Prozessinstanz ändern. Ändern Sie Variablen beispielsweise, um die mögliche Fehlerursache eines Prozesses zu beheben.

Wählen Sie **Variablen bearbeiten** aus, um den Bearbeitungsmodus zu aktivieren und die Werte in den Eingabefeldern zu ändern. Sie haben zwei Möglichkeiten, den Bearbeitungsmodus zu verlassen:

- Wählen Sie **Änderungen speichern** aus, um die neuen Werte in die Datenbank zu schreiben und den Bearbeitungsmodus zu verlassen. Alle Änderungen haben eine direkte Auswirkung auf den gesamten Prozess.
- Wählen Sie **Änderungen verwerfen** aus, um die alten Ausgangswerte wiederherzustellen und den Bearbeitungsmodus zu verlassen. Dieser Vorgang hat keine Auswirkungen auf den Prozess.

### 1.3.5. Erneutes Ausführen eines fehlgeschlagenen Tokens

Wenn Sie im Feature **Prozessüberwachung** einen Prozess im Status **Fehler** sehen, können Sie den fehlerhaften Token zur Fehlerbehebung erneut ausführen. Beachten Sie die Fehlerursache in der Detailansicht

eines Tokens, da Sie für die Fehlerbehebung möglicherweise die Variablen der Prozessinstanz ändern müssen.

Zum erneuten Ausführen eines Tokens navigieren Sie in die Detailansicht des Tokens. Wählen Sie **Erneut versuchen** aus und bestätigen Sie den Hinweis.

## 1.3.6. Erneutes Ausführen eines asynchronen Services

Bei der Verwendung eines asynchronen Services kann es beim Ausführen der Serviceaktivität zu einem Fehler bei der Kommunikation mit dem externen Dienst kommen. Der Fehler kann auftreten, wenn der Service-Endpunkt die Serviceanfrage zwar entgegennimmt, aber aufgrund von technischen Problemen nicht darauf antwortet. Um diesen Fehler zu beheben, können Sie in der Prozessüberwachung einen erneuten Aufruf des Services ausführen.

**So geht's**

1. Öffnen Sie im Feature **Prozessüberwachung** die betroffene Prozessinstanz und das Service-Token, das nicht beantwortet wurde.
2. Wählen Sie **Token verschieben** aus.
3. Wählen Sie die Zielaktivität für das Token aus und bestätigen Sie den Dialog.

## 1.3.7. Wissenswertes zur Protokollierung von Prozessen

Um den Durchlauf einzelner Prozessinstanzen nachverfolgen zu können, können Sie für die zugehörige Prozessversion die Protokollierung aktivieren.

Sie haben zwei Möglichkeiten, um das Protokoll anzuzeigen. In der Prozessüberwachung können Sie das Protokoll mit der Kontextaktion **Prozessprotokoll** anzeigen. Für Aufgaben, die Teil eines protokollierten Prozesses sind, können Sie die Protokollierung in der Perspektive **Prozess** in der Aufgabenübersicht anzeigen.

**Unterschiedliche Ansichten**

Für die Prozessprotokolle gibt es zwei verschiedene Ansichten:

- In der **fachlichen Ansicht** wird der Fokus auf Benutzeraktivitäten gelegt. Das Protokoll enthält in der fachlichen Ansicht Informationen zum Prozessstart und Prozessende, zu Benutzeraktivitäten, zu beteiligten Personen sowie zu eingegebenen Daten.
- In der **technischen Ansicht** werden alle durchlaufenen Aktivitäten der Prozessinstanz aufbereitet. Dazu zählen alle Elemente, die in der BPMN-Definition vorliegen, Serviceaufrufe, eingetretene Fehlersituationen und viele weitere technische Details. Die technische Ansicht eines Protokolls ist nur in der Prozessüberwachung verfügbar.

**Aufbewahrung von Protokollen**

Protokolle sind während der gesamten Laufzeit einer Prozessinstanz verfügbar. Nach Abschluss der Prozessinstanz können die Protokolle abhängig von der Konfiguration bis zu einem Jahr aufbewahrt werden. Nach Ablauf der Aufbewahrungsfrist werden die Protokolldaten gelöscht.

**Export von Protokollen in ein d.3-Repository**

Wenn Sie Protokolldaten über die Aufbewahrungsfrist hinaus aufbewahren möchten, können Sie die Protokolle in Ihrem d.3-Repository speichern. Zum Speichern der Protokolle im Repository müssen Sie zunächst den Protokollexport konfigurieren. Nach Abschluss einer Prozessinstanz wird das Protokoll anschließend automatisch in Ihrem d.3-Repository gespeichert.

Sie können die Protokolle nach dem Export weiterhin in der Prozessüberwachung und in der Aufgabenansicht anzeigen.

**Besondere Prozessvariablen**

Sie können dem Protokoll bei Benutzeraufgaben zusätzliche Informationen in Form einer Entscheidung und eines Kommentars hinzufügen. Dafür können Sie die Prozessvariablen **dv_decision** und **dv_comment** durch einen Bearbeitungsdialog übergeben.

Die Inhalte werden im Protokoll besonders hervorgehoben.

### 1.3.8. Aktivieren der Protokollierung für eine Prozessdefinition

Beim Bereitstellen einer Prozessdefinition haben Sie die Möglichkeit, die Protokollierung zu aktivieren. Standardmäßig werden Protokolle für 30 Tage ab dem Prozessende aufbewahrt.

Sie müssen die Protokollierung für jeden Prozess einzeln einstellen.

**So geht's**

1. Öffnen Sie das Feature **Prozessadministration**.
2. Wähle Sie die Aktion **Neuen Prozess bereitstellen** aus.
3. Wählen Sie die BPMN-Datei aus, die Sie bereitstellen möchten.
4. Klicken Sie auf **Hochladen**.
5. Klicken Sie auf **Prozessprotokollierung**, um die Einstellungen für die Protokollierung zu öffnen.
6. Aktivieren Sie die Protokollierung.
7. Aktivieren Sie optional den Export für Protokolle in Ihr d.3-Repository, um eine revisionssichere Archivierung zu verwenden.
8. Ändern Sie bei Bedarf die Aufbewahrungszeit der Protokolle.
9. Bestätigen Sie den Dialog mit **Bereitstellen**.

Wenn Sie eine neue Version einer Prozessdefinition bereitstellen, werden die zuletzt verwendeten Einstellungen für die Protokollierung der Prozessversion übernommen. Wenn Sie die Einstellungen nachträglich ändern möchten, müssen Sie die Prozessdefinition erneut bereitstellen.

### 1.3.9. Erstellen und konfigurieren eines Prozessereignisses

Mithilfe von Prozessereignissen können Sie Prozesse über einen JSON-Endpunkt (z.B. via Webhook) starten. Erstellen Sie dafür zunächst ein Prozessereignis.

**So geht's**

1. Öffnen Sie das Feature **Prozessadministration**.
2. Wähle Sie **Prozessereignisse** aus.
3. Klicken Sie auf das Plussymbol, um ein neues Prozessereignis zu erstellen.
4. Geben Sie einen Namen für das zu erstellende Prozessereignis ein und klicken Sie auf **Hinzufügen**. Das Prozessereignis ist wird erstellt. Sie müssen dieses nun konfigurieren.
5. Wählen Sie im Auswahlfeld **Prozess** den Prozess aus, der gestartet werden soll.
6. Sie können optional **Business Key** oder **Correlation Key** für den Prozessstart definieren. Geben Sie hierzu im entsprechenden Feld einen Ausdruck an.
7. Im Bereich **Zuordnungen definieren** können Sie Daten aus dem JSON-Endpunkt zu Prozessvariablen zuordnen.
8. Klicken Sie auf das Plussymbol, um eine neue Zuordnung zu definieren.
9. Geben Sie einen Namen für die Zuordnung ein.
10. Geben Sie im Feld **Endpunktpfad** einen Ausdruck an, dessen Ergebnis beim Start in die Prozessvariable geschrieben wird.
11. Wählen Sie die **Prozessvariable** aus.
12. Klicken Sie auf **Hinzufügen**, um die Zuordnung durchzuführen.
13. Wiederholen Sie die Punkte 8-12 für alle Zuordnungen, die Sie definieren möchten. Beachten Sie, dass alle Pflichtvariablen des Prozesses zugeordnet werden müssen.
14. Speichern Sie das Prozessereignis.

In der Übersicht der Prozessereignisse können Sie die URL zum Auslösen eines Prozessereignisses jeweils im Drei-Punkte-Menü durch Klick auf **Endpunkt-URL in Zwischenablage kopieren** ermitteln.

Weitere Informationen zum Aufruf des Endpunktes finden Sie in der API-Dokumentation.

### 1.3.10. Wissenswertes zu Ausdrücken in Prozessereignissen

Sie können beim Erstellen von Prozessereignissen variable Ausdrücke für die Zuordnung von Werten aus dem JSON des Endpunktaufrufes verwenden. Die Ausdrücke verwenden die Syntax der Java Expression Language.

Ausdrücke dürfen lediglich folgende Elemente verwenden:

| Element | Restriktion |
|---|---|
| Lesender Zugriff auf die Eigenschaften aus dem beim Aufruf übermittelten JSON**: input.getValue** | Der an die Methode übergebene String muss einem gültigen JSON-Path entsprechen (siehe Beispiele). |
| Erzeugen von Mehrfachwerten: **collections.from** | |
| Operatoren und Zeichen | Erlaubte Operatoren und Zeichen: +, -, *, /, %, =, !, <, >, &, \|, ?, (, ), [, ]. |
| Keyword | Erlaubte Keywords: div, mod, eq, ne, lt, gt, le, ge, and, or, not, empty. |
| Numerische Werte | |
| Zeichenketten | Zeichenketten müssen in " " oder ' ' eingeschlossen sein. |

**Beispiele**

Beim Endpunktaufruf übermitteltes JSON


```
{
   "doc":{
      "caption":"Process description - 2021-11-17",
      "categoryId":"dv.fol.basis.Correspondence",
      "dateCreated":"2021-11-17T07:39:36.502+00:00",
      "dateLastAccess":"2021-11-17T07:39:36.502+00:00",
      "dateUpdated":"2021-11-17T07:39:36.502+00:00",
      "dateUpdatedFile":"2021-11-17T07:39:36.502+00:00",
      "fileExtension":"PDF",
      "fileName":"correspondence",
      "id":"KR00000227",
      "number":"KR00000227",
      "size":51500,
      "status":"F",
      "versions":[
         {
            "id":0,
            "status":"F",
            "dateCreated":"2021-11-17T07:39:37.000+00:00",
            "fileName":"KR00000227.1",
            "fileExtension":"PDF",
            "size":51500
         }
      ],
      "properties":[
         {
            "id":"dv.fol.basis.Subject",
            "name":"Subject",
            "index":2,
            "dataType":"STRING",
            "isMultiValue":false,
```

```
            "value":"Process description"
         },
         {
            "id":"dv.fol.basis.Date",
            "name":"Date",
            "index":50,
            "dataType":"DATE",
            "isMultiValue":false,
            "value":"2021-11-17"
         }
      ]
   },
   "fileDestination":"somwhere",
   "importOk":"1",
   "user":{
      "id":"A64C2BD6-8EF5-4C6D-B1BA-7F632A2AC3ED",
      "name":"Jane Doe",
      "d3Id":"jdoe"
   },
   "docType":{
      "id":"dv.fol.basis.Correspondence",
      "name":"Correspondence",
      "type":"DOCUMENT_TYPE"
   }
}
```

**Beispiele**

```
// Hello World
${"Hello World"}

// 3
${3}

// false
${3 > 4}

// Value of doc.status in JSON
${input.getValue("$.doc.status")}

// Value of a defined element in the doc properties in JSON.
// If an element is found, an array with one element is returned. Can NOT
be used in combination with string concatenation.
${input.getValue("$.doc.properties[?(@.id ==
\'dv.fol.basis.Subject\')].value")}

// Value of a defined element in the doc properties in JSON.
// If an element is found, its value is returned and may be used within
string concatenation. With this expression you have to ensure, that an
element exists.
${input.getValue("$.doc.properties[?(@.id ==
\'dv.fol.basis.Subject\')].value")[0]}

// creates a collection with "a", "b" and "c"
${collections.from("a", "b", "c")}
```

Es sind auch Mischformen möglich, sodass Sie Texte und variable Inhalte direkt miteinander kombinieren können.

```
dmsObject:///dms/r/yourRepositoryId/o2/${input.getValue("$.doc.id")}
```

# 1.4. Modellieren von Prozessen

In diesem Thema erfahren Sie, wie Sie Ihre eigenen Prozesse modellieren. Sie lernen die unterstützten BPMN-Elemente kennen und erfahren, wie Sie diese zu Ihren Prozessen hinzufügen.

## 1.4.1. Wissenswertes zur Prozessmodellierung

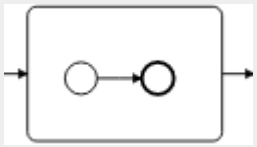
Bei der Erstellung von Prozessmodellen unterstützt die Anwendung den BPMN-Standard. Dabei handelt es sich um ein XML-basiertes Format. Das Modellieren von Prozessen im d.velop-Kontext orientiert sich am Standard der Object Management Group (OMG). Der Wert **http://www.omg.org/spec/BPMN/20100524/MODEL** ist der standardmäßige Namensraum.

Für die Modellierung von Prozessmodellen empfehlen wir die Verwendung von d.velop process modeler.

## 1.4.2. Verwenden von BPMN-Elementen

Bei der Erstellung eigener Prozessmodelle können Sie folgende BPMN-Elemente des BPMN-Standards 2.0 verwenden:

| | BPMN-Element | Restriktionen |
|---|---|---|
|  | **Start event** (Startereignis) | Erlaubte Ereignistypen: None, Timer<br><br>Nur in eventbasierten Subprozessen: Eskalation, Error |
|  | **End event** (Endereignis) | Erlaubte Ereignistypen: Keine, Terminierung, Eskalation, Error. |
|  <br>  <br>  | **Intermediate boundary event** (angeheftetes Zwischenereignis) | Erlaubte Ereignistypen: Timer, Eskalation, Error. |
|  | **Intermediate catch event** (Zwischenereignis mit wartender Funktion) | Erlaubte Ereignistypen: Timer. |
|  | **Intermediate throw event** (Zwischenereignis mit auslösender Funktion) | Erlaubte Ereignistypen: Eskalation. |
|  | **User Task** (Benutzeraktivität) | |
|  | **Send Task** (Sendenaktivität) | Nur zur Anbindung von Services. |
|  | **Receive Task** (Empfangenaktivität) | Nur zur Anbindung von Services. |

| | BPMN-Element | Restriktionen |
|---|---|---|
|  | **Service Task** (Serviceaktivität) | Nur definierte Aktionen (**writeLocalVariables**, **writeGlobalVariable** oder **writerMultiInstanceVariables**) erlaubt. |
|  | **Gateway** (Verzweigung) | Erlaubte Verzweigungstypen: Exclusiv, Parallel, Inclusiv. |
|  | **Sequence flow** (Sequenzfluss) | |
|  | **Sub process** (Subprozesse) | Erlaubte Subprozesstypen: Normal, eventbasiert (s. Startereignis) |
|  | **Pool/Participant/Lane** | Pro Prozessdiagram ist maximal ein Pool bzw. Participant erlaubt. |

**Allgemeine Einschränkungen**

Für alle Elemente gelten folgende Restriktionen:

- Ausdrücke bei In- und Output-Parametern unterliegen den Restriktionen für Ausdrücke.
- Es sind ausschließlich die folgenden camunda-Erweiterungen erlaubt:
  - **camunda:inputOutput**
  - **camunda:properties**
  - **camunda:property**
  - **camunda:failedJobRetryTimeCycle**
  - **camunda:asyncBefore**
  - **camunda:asyncAfter**
  - **camunda:formKey**

## 1.4.3. Erstellen eines exemplarischen Prozesses in einer BPMN-Datei

Für den Einstieg in das Modellieren eigener Prozesse ist es hilfreich, wenn Sie zunächst einen einfachen Prozess beispielhaft erstellen. Voraussetzung für das Modellieren eigener Prozesse sind eingerichtete Benutzerrollen. Außerdem benötigen Sie einen Text- oder BPMN-Editor mit XML-Ansicht und ein Tool zum Aufrufen einer REST-API.

Ein einfacher Prozess sieht beispielsweise so aus:


Erstellen Sie zum Modellieren Ihres eigenen Prozesses zunächst folgendes Grundgerüst in Ihrer BPMN-Datei:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process>
    <startEvent />
    <sequenceFlow />
    <userTask />
    <sequenceFlow />
    <endEvent />
  </process>
</definitions>
```

Diesem Grundgerüst fügen Sie die einzelnen Prozesselemente hinzu.

**Hinzufügen des Prozesselements**

Das BPMN-Element **process** ist eine Kapsel um den gesamten Prozess. Fügen Sie dem BPMN-Element die Eigenschaften **id**, **name** und **isExcecutable** hinzu.

```
<process id="hello_world_process" name="Hello World Process"
isExecutable="true">
```

**Erläuterung der Eigenschaften**

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für den Prozess verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **isExecutable**: Die Eigenschaft muss den Wert **true** enthalten, damit der Prozess gestartet werden kann.

**Hinzufügen eines Start- und Endereignisses**

Die BPMN-Elemente **startEvent** und **endEvent** werden als Markierungen verwendet, die anzeigen, wo der Prozess beginnt und wo er endet. Fügen Sie den BPMN-Elementen jeweils die Eigenschaft **id** hinzu.

```
<startEvent id="start" />
<endEvent id="end" />
```

**Hinzufügen einer Benutzeraktivität**

Das BPMN-Element **userTask** stellt eine Benutzeraktivität im Prozess dar. Bei Eingang in diese Aktivität wird einem oder mehreren Benutzern eine Aufgabe zugestellt. Fügen Sie dem BPMN-Element die Eigenschaften **id**, **name** und **camunda:candidateUsers** hinzu.

```
<userTask id="task_hello_world" name="Hello World" camunda:candidateUsers="$
{variables.get('dv_initiator')}" />
```

**Erläuterung der Eigenschaften**

- **id**: Die Eigenschaft dient als Bezeichner der Aktivität im BPMN-Modell und wird bei der Verbindung durch Sequenzflüsse verwendet.
- **name**: Die Eigenschaft wird der Aufgabe als Betreff hinzugefügt.
- **camunda:candidateUsers**: Die Eigenschaft bezieht sich auf die Person, die die Aufgabe bearbeiten soll. In der BPMN-Datei wird mit dem Ausdruck **${variables.get("dv_initiator")}** eine Variable verwendet. Die Variable **dv_initiator** verweist immer auf die Person, die den Prozess gestartet hat. Diese Aufgabe wird also immer der Person zugestellt, die den Prozess startet.

**Hinzufügen von Sequenzflüssen**

Verbinden Sie die einzelnen Prozesselemente, um einen Ablauf zu erstellen. Zum Verbinden der Prozesselemente verwenden Sie das BPMN-Element **sequenceFlow**. Fügen Sie dem BPMN-Element die Eigenschaften **id**, **sourceRef** und **targetRef** hinzu.

Die beispielhafte Konfiguration in der BPMN-Datei zeigt, wie Sie das Startereignis mit der Benutzeraktivität verbinden und die Benutzeraktivität mit dem Endereignis verknüpfen:

```
<sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
<sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
```

**Erläuterung der Eigenschaften**

- **id**: Die Eigenschaft ist ein eindeutiger Bezeichner für den Sequenzfluss innerhalb des BPMN-Modells.
- **sourceRef**: Die Eigenschaft verweist auf den Ausgangspunkt des Sequenzflusses.
- **targetRef**: Die Eigenschaft verweist auf das Ziel des Sequenzflusses. In dieser Eigenschaft tragen Sie die Bezeichner der zu verbindenden Elemente ein.

Nachdem Sie die einzelnen Prozesselemente hinzugefügt haben, sieht die BPMN-Datei folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="hello_world_process" name="Hello World Process"
isExecutable="true">
    <startEvent id="start" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world"
/>
        <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <endEvent id="end" />
  </process>
</definitions>
```

**Hinzufügen von grafischen Informationen**

Wenn Sie Ihren erstellten Prozess in einem Diagramm anzeigen möchten, können Sie der BPMN-Datei grafische Informationen hinzufügen. Diese grafischen Informationen legen die Größe des Diagramms und die Position der einzelnen Elemente fest. Die grafischen Informationen fügen Sie mithilfe des BPMN-Elements **bmpndi:BPMNDiagram** hinzu.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
```

```
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
  <process id="hello_world_process" name="Hello World Process"
isExecutable="true">
    <startEvent id="start" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <endEvent id="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane" bpmnElement="hello_world_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="173" y="102" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="209" y="120" />
        <di:waypoint x="259" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="259" y="80" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="359" y="120" />
        <di:waypoint x="409" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="409" y="102" width="36" height="36" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

### 1.4.4. Verwenden von Ausdrücken

Sie können beim Modellieren von Prozessen variable Ausdrücke verwenden. Die Ausdrücke verwenden die Syntax der Java Expression Language.

Ausdrücke dürfen lediglich folgende Elemente verwenden:

| Element | Restriktion |
|---|---|
| Lesender Zugriff auf die Variablen mittels **variables**<br><br>• **variables.get**<br>• **variables.getDisplayValue**<br>• **variables.getObjectValue** | |
| Erzeugen von Mehrfachwerten<br><br>• **collections.from** | |

| Element | Restriktion |
|---|---|
| Lesender Zugriff auf Werte der aktuellen Prozessinstanz<br><br>• **process.instance.id**<br>  • ID der aktuellen Prozessinstanz<br>• **process.instance.businessKey**<br>  • Business-Key der aktuellen Prozessinstanz | |
| Operatoren und Zeichen | Erlaubte Operatoren und Zeichen: +, -, *, /, %, =, !, <, >, &, \|, ?, (, ), [, ]. |
| Keyword | Erlaubte Keywords: div, mod, eq, ne, lt, gt, le, ge, and, or, not, empty. |
| Numerische Werte | |
| Zeichenketten | Zeichenketten müssen in " " oder ' ' eingeschlossen sein. |

**Beispiele**

```
${"Hello World"}                              // Hello World
${3}                                          // 3
${3 > 4}                                      // false
${variables.get("myVariable")}                // Value of myVariable
${variables.get("myVariable") == "Hello"}     // true, if myVariable equals
Hello, false otherwise
${collections.from("a", "b", "c")}            // creates a collection with
"a", "b" and "c"
```

Es sind auch Mischformen möglich, sodass Sie Texte und variable Inhalte direkt miteinander kombinieren können.

**Beispiele**

```
This is an expression with ${variables.get("myVariable")}
3 is ${(3 > 4) ? "greater" : "lesser"} than 4
```

## 1.4.5. Arbeiten mit Prozessvariablen

In diesem Thema erfahren Sie, wie Sie Ihre Prozesse mithilfe von Variablen dynamisch erstellen.

## Wissenswertes zu Prozessvariablen

Prozessvariablen sind Daten, die Informationen zu einer Prozessinstanz enthalten. Folgende Prozessvariablen sind in jeder Prozessinstanz vorhanden:

- **dv_initiator**: Eine Prozessvariable vom Datentyp **Identity**. Die Prozessvariable verweist auf die Identität des Benutzers, der die Prozessinstanz gestartet hat.
- **dv_start_time**: Eine Prozessvariable vom Datentyp **String**. Die Prozessvariable enthält den UTC-String des Zeitpunktes, in dem die Prozessinstanz aus Sicht der Anwendung gestartet wurde. Es gilt der Zeitpunkt, in dem die Anwendung beauftragt wurde, die Prozessinstanz zu starten. Da der eigentliche (technische) Start der Instanz asynchron durchgeführt wird, kann der in der Variable enthaltene Zeitpunkt möglicherweise abweichen.

**Verwenden weiterer Prozessvariablen**

Diese Variablen können Sie für spezielle Zwecke verwenden:

- **dv_sender**: Eine Prozessvariable vom Datentyp **Identity**. Die Prozessvariable verweist auf die Identität des Benutzers, der als Absender von Benutzeraktivitäten verwendet wird.
- **dv_attachment**: Eine Prozessvariable vom Datentyp **DmsObject** oder abweichend. Die Prozessvariable enthält eine Verknüpfung, die allen Benutzeraktivitäten als Anhang hinzugefügt wird.
- **dv_decision:** Eine Prozessvariable vom Typ **String**. Dieser Text wird im Prozessprotokoll bei einer Aufgabe als **Entscheidung** hervorgehoben.
- **dv_comment:** Eine Prozessvariable vom Typ **String**. Dieser Text wird im Prozessprotokoll bei einer Aufgabe als **Kommentar** hervorgehoben.

### Warnung

Die Prozessvariable **dv_attachment** ist mit dem Datentyp **DmsObject** vordefiniert. Möchten Sie eine andere URI als die zu einem Objekt der DMS-App verwenden, müssen Sie den Wert des Datentyps auf **String** ändern.

## Definieren von Prozessvariablen

Definieren Sie in der Prozessdefinition Variablen, die innerhalb eines Prozesses genutzt werden. Definierte Variablen können an den entsprechenden Schnittstellen der Anwendung automatisch validiert und ggf. konvertiert werden.

Innerhalb eines Prozesses können Sie folgende Variablen definieren:

- **Allgemeine Prozessvariable**: Eine Variable, die im gesamten Prozess Gültigkeit hat. Sie können festlegen, ob es sich um eine Startvariable handelt.
- **Lokale Prozessvariable**: Eine Variable, deren Gültigkeitsbereich auf eine einzelne Prozessaktivität beschränkt ist.
- **Servicevariablen**: Ein- oder Ausgabevariablen eines Prozessservices.

Eine Variablendefinition beinhaltet folgende Informationen:

| Eigenschaft | Mögliche Werte | Beispiel |
|---|---|---|
| Name | [A-Za-z][A-Za-z0-9_]* | myVariable (**name**-Eigenschaft) |
| Startvariable | | myStartVariable* (**name**-Eigenschaft) |
| Datentyp | • String<br>• Number<br>• Identity<br>• DmsObject<br>• URL<br>• Object<br>• File | String (**value**-Eigenschaft) |
| Einzel- oder Mehrfachwert | | Einzelwert: String (**value**-Eigenschaft)<br>Mehrfachwert: [String] (**value**-Eigenschaft) |
| Pflichtvariable | | Einzelwert: String! (**value**-Eigenschaft)<br>Mehrfachwert: [String]! (**value**-Eigenschaft) |

### Anmerkung

Besonderheiten des Datentyps **Object**:

- Sie können in einem Prozess nur eine Variable vom Typ **Object** definieren.
- Eine Variable vom Typ **Object** darf keinen Mehrfachwert enthalten.
- Variablen vom Typ **Object** werden nicht protokolliert.

### Anmerkung

Besonderheiten des Datentyps **File**:

- Eine Datei darf maximal 100 MB groß sein.
- Die Gesamtgröße aller Dateien einer Prozessinstanz darf 500 MB nicht überschreiten.
- Dateien werden nach Prozessende gelöscht.
- Variablen vom Typ **File** werden nicht protokolliert.

### Warnung

Beachten Sie beim Verwenden von Prozessvariablen folgende wichtige Hinweise:

- **Namenspräfix "dv_"**: Sie dürfen für Variablen das Namenspräfix **dv_** nicht verwenden. Diese Variablen sind für interne Variablen reserviert. Sie dürfen das Namenspräfix **dv_** lediglich beim Überschreiben der Variablendefinition **dv_attachment** verwenden.
- **Ändern von Variablendefinitionen beim Versionswechsel**: Wenn Sie zwischen verschiedenen Versionen eines Prozesses den Datentypen einer Variablen ändern oder zwischen Einzelwert und Mehrfachwert wechseln, kann es nach der Migration zu Fehlern in der Prozessausführung kommen. Die Fehler treten auf, wenn bestehende Prozessinstanzen für die entsprechende Variable bereits Werte der bisherigen Definition beinhalten. Nach einer Migration in die neue Version verlieren diese Werte ihre Gültigkeit. Stellen Sie bei Änderungen der Variablendefinitionen immer sicher, dass keine zu migrierende Prozessinstanz betroffen ist.

### Anmerkung

Sie können eine Variable, die als Pflichtvariable definiert ist, mit der Schnittstelle zum Ändern von Variablen nicht löschen. Handelt es sich bei der Pflichtvariable zusätzlich um eine Startvariable, muss ein Anwender beim Start einer Prozessinstanz immer einen gültigen Wert für die Variable übergeben. Wenn er keinen gültigen Wert übergibt, wird die Anfrage mit dem Fehlercode 400 abgelehnt.

Für den Datentypen **Number** gelten diese Einschränkungen:

| Minimum | Maximum | Nachkommastellen | Genauigkeit |
|---|---|---|---|
| -1e16 | 1e16 | 5 | 15 Stellen |

Die nicht-primitiven Datentypen verwenden die folgende Notation:

| Datentyp | Notation |
|---|---|
| **Identity** | **Für Benutzer:** identity:///identityprovider/scim/users/<someUserId><br>**Für Gruppen:** identity:///identityprovider/scim/groups/<someGroupId> |
| **DmsObject** | dmsObject:///dms/r/<repositoryId>/o2/<dmsObjectId> |
| **URL** | https://www.d-velop.de **(max. 2000 Zeichen)** |
| **Object** | { "myKey" : "myValue"} **(JSON-String, max. 50 KB)** |

Variablen werden als BPMN-Erweiterung in Form von Camunda-Properties definiert. Allgemeine Prozess- und Servicevariablen definieren Sie innerhalb des BPMN-Modells direkt im Prozessknoten. Lokale Prozessvariablen definieren Sie im Knoten der Aktivität, in der sie gültig sein sollen.

#### Allgemeine Prozess- und Servicevariablen

```
 <process id="myProcess" name="Process with variable declaration"
isExecutable="true">
    <extensionElements>
        <camunda:properties>
            <camunda:property name="variable:myVariable" value="[String]!"
/> <!-- Name: "myVariable", Type: "String", Multiple: true, Mandatory:
true, Startvariable: false -->
            <camunda:property name="variable:myStartVariable*"
value="String" /> <!-- Name: "myStartVariable", Type: "String", Multiple:
false, Mandatory: false, Startvariable: true-->
            <camunda:property name="variable:someUserTaskOutput"
```

```
value="String" /> <!-- Name: "someUserTaskOutput", Type: "String",
Multiple: false, Mandatory: false, Startvariable: false -->
            <camunda:property name="service:/myservice:in:input"
value="String!" /> <!-- Name: "input", Type: "String", Multiple: false,
Mandatory: true -->
            <camunda:property name="service:/myservice:out:output"
value="Number!" /> <!-- Name: "output", Type: "Number", Multiple: false,
Mandatory: true -->
        </camunda:properties>
    </extensionElements>
</userTask>
```

**Lokale Prozessvariable**

```
<userTask id="userTask" name="User Task">
    <extensionElements>
        <camunda:inputOutput>
            <camunda:inputParameter name="someInput">user task input which
is only available within this user task scope</camunda:inputParameter>
            <camunda:outputParameter name="someUserTaskOutput">user task
output which will be written to process scope</camunda:outputParameter>
        </camunda:inputOutput>
        <camunda:properties>
            <camunda:property name="variable:someInput" value="String" />
        </camunda:properties>
    </extensionElements>
</userTask>
```

## Verwenden von Prozessvariablen

Beim Modellieren eigener Prozesse können Sie Prozessvariablen verwenden, um dynamische Prozesse zu erstellen. Zum Einstieg in das Arbeiten mit Prozessvariablen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Dieser Prozess sieht beispielsweise so aus:


Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut. Zur Vereinfachung sind die grafischen Informationen zum BPMN-Diagramm in dieser BPMN-Beispieldatei nicht enthalten.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="GettingStarted" name="Beispiel: Einstieg"
isExecutable="true">
    <startEvent id="start"/>
    <userTask id='task_hello_world' name='Hello World'
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
        <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end"
```

```
/>
    <endEvent id="end"/>
  </process>
</definitions>
```

Dieser Prozessdefinition fügen Sie nun eine neue Variablendefinition hinzu.

**Hinzufügen einer Variablendefinition**

Um eine Prozessvariable verwenden zu können, müssen Sie diese der Prozessdefinition hinzufügen. In der Prozessdefinition fügen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** zum BPMN-Element **process** hinzu. Zum Definieren der Prozessvariable fügen Sie dem BPMN-Element **camunda:properties** noch das Element **camunda:property** hinzu.

Dieses Beispiel zeigt, wie Sie die Variable **message** vom Datentyp **String** hinzufügen:

```
<process id="GettingStarted" name="Beispiel: Einstieg" isExecutable="true">
        <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:message" value="String" />
      </camunda:properties>
    </extensionElements>
        ...
</process>
```

**Erläuterung der Eigenschaften**

- **name**: Die Eigenschaft definiert den Namen der Prozessvariable. Vor dem Wert steht immer das Präfix **variable:**.
- **value**: Die Eigenschaft definiert den Datentypen der Prozessvariable.

**Verwenden der Prozessvariable**

Sie können die Variable **message** nun für den Namen der Benutzeraktivität verwenden, damit dieser Name dem Anwender in der Aufgabenliste angezeigt wird. Hierzu ersetzen Sie den vorhandenen Text in der Eigenschaft **name** der Benutzeraktivität durch die Methode für den Zugriff auf die Variable **message**:

```
<process id="GettingStarted" name="Beispiel: Einstieg" isExecutable="true">
        ...
        <userTask id="task_hello_world" name="${variables.get('message')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
        ...
```

Sie können den Wert der Variablen nun zum Beispiel beim Start des Prozesses belegen.

## Ermitteln von Prozessvariablen

Mit dem Objekt **variables** können Sie auf die Prozessvariablen zugreifen. Der Aufruf erfolgt jeweils in der Notation **variables.<methodName>( <Parameter>)**.

Sie können die Methoden nicht auf Prozessvariablen vom Typ **File** anwenden.

**get( String variableName)**

Die Methode ermittelt den Wert für die Variable mit dem eingegebenen Variablennamen. Bei Variablen mit einem Mehrfachwert werden die Werte in einer Liste (collection) zurückgegeben.

**getDisplayValue( String variableName)**

Die Methode ermittelt den Wert für die Variable mit dem eingegebenen Variablennamen. Je nach Datentyp findet hierbei automatisch eine entsprechende Konvertierung statt.

Bei Variablen mit einem Mehrfachwert werden die konvertierten Werte in einem String mit Kommas getrennt zurückgegeben. Sind die Variablen vom Datentyp **Number**, werden die konvertierten Werte in einem String mit Semikolons getrennt zurückgegeben.

| Datentyp | Tatsächlicher Wert | Rückgabewert |
|---|---|---|
| String | "some value" | "some value" |
| Number | 1.23 | "1.23" |
| Identity | identity:///identityprovider/scim/users/userIdOf-MaxMustermann | "Max Mustermann" (**displayName**-Eigenschaft des Benutzers, **name**-Eigenschaft bei Gruppen im d.ecs identity provider) |
| DmsObject | dmsObject:///dms/r/someRepo/o2/T000000001 | "/dms/r/someRepo/o2/T000000001" |
| Object | {"myKey":"myValue"} | "{\"myKey\":\"myValue\"}" (JSON-String) |

**getObjectValue( String variableName, String objectPath)**

Diese Methode kann nur für Variablen vom Typ **Object** verwendet werden. Sie ermöglicht es, auf einzelne Teile des Objekts zuzugreifen.

Beispiel:

Angenommen, es gibt eine Variable vom Typ **Object** mit dem Namen "myObjectVariable" und folgendem Wert:

```
{
  "my" : {
    "object" : {
      "path" : 123
    }
  }
}
```

Dann können mit der Methode **getObjectValue** z. B. die folgenden Teile des Objekts ermittelt werden:

| Methodenaufruf | Rückgabewert |
|---|---|
| getObjectValue("myObjectVariable", "my.object.path") | 123 |
| getObjectValue("myObjectVariable", "my.object") | {"path":123} |

## Erzeugen von Mehrfachwerten

Mit dem Objekt **collections** können Sie aus Einzelwerten eine Liste (collection) erzeugen, z.B. um einer Variablen einen Mehrfachwert zuzuweisen. Das Erzeugen einer Liste aus Einzelwerten ist beispielsweise hilfreich, wenn Sie in einer Benutzeraktivität mehrere Empfänger als Konstante hinterlegen wollen.

**Beispiel**

```
${collections.from( "identity:///identityprovider/scim/users/user1",
"identity:///identityprovider/scim/users/user")}
```

**from( Object...objects)**

Die Methode erzeugt eine Liste (collection) mit den übergebenen Einzelwerten.

## Wissenswertes zum Geltungsbereich von Prozessvariablen

Prozessvariablen haben innerhalb einer Prozessinstanz immer einen bestimmten Geltungsbereich. Je nach Geltungsbereich sind lesende und schreibende Zugriffe möglich.

In jeder Prozessinstanz gibt es einen universellen Geltungsbereich, der den gesamten Prozess umfasst. Alle weiteren Geltungsbereiche, wie zum Beispiel Benutzer- oder Serviceaktivitäten, können lesend auf die Variablen im universellen Geltungsbereich zugreifen.

Zusätzliche Geltungsbereiche werden an folgenden Stellen erzeugt:

- Verzweigungen mit parallelen Pfaden
- Multi-Instanz-Konstrukte
- Subprozesse
- Input-/Output-Mapping oder Timer-Ereignis bei Benutzeraktivitäten
- Serviceaktivitäten

**Lesende Zugriffe**

Lesende Zugriffe auf Prozessvariablen (z.B. die Verwendung in einem Ausdruck) sind unabhängig vom Geltungsbereich möglich. Wird im aktuellen Geltungsbereich kein Wert gefunden, so wird der darüber liegende Bereich durchsucht, bis ein Wert gefunden wurde.

**Schreibende Zugriffe**

Wenn Sie Werte ändern möchten (z.B. in einer Benutzeraktivität mit Output-Mapping), so wird der Wert immer zunächst in den aktuellen Geltungsbereich der Aktivität übernommen. Andere Geltungsbereiche, z.B. entlang anderer Zweige im Prozess, werden dadurch bis zum Ende dieses Geltungsbereiches nicht beeinflusst. Erst am Ende des aktuellen Wertebereichs werden die Änderungen auch in alle anderen Geltungsbereiche übernommen.

**Ende von Geltungsbereichen**

Am Ende eines Geltungsbereichs (z.B. bei der Zusammenführung paralleler Zweige oder am Ende eines Subprozesses) werden die Variablen dieses Geltungsbereichs verworfen. Um Variablen in einen darüber liegenden Geltungsbereich zu übernehmen, müssen Sie ein Output-Mapping konfigurieren. Dabei wird die Aufrufhierarchie vom aktuellen Element ausgehend so lange nach oben durchlaufen, bis ein Geltungsbereich gefunden wird, in dem diese Variable bereits existiert. In diesen wird der neue Wert dann übernommen. Für den Fall, dass es sich um eine komplett neue Variable handelt, wird diese dann in den Geltungsbereich des Gesamtprozesses geschrieben.

> **Warnung**
> Bitte beachten Sie, dass ein Timer-Ereignis bei Benutzeraktivitäten einen eigenen Geltungsbereich erzeugt. Wenn Sie kein dediziertes Input-/Output-Mapping für die Benutzeraktivität definieren, werden bei Abschluss der Aktivität alle Variablen dieses Geltungsbereiches in den höheren Geltungsbereich übernommen.

**Auswirkungen in der Prozessüberwachung**

Bei der Ansicht eines Tokens werden immer die Variablen des aktuellen Geltungsbereichs angezeigt. Änderungen an den Variablen werden nur in diesen Geltungsbereich übernommen.

Die meisten Tokens besitzen einen eigenen Geltungsbereich. Die einzige Ausnahme bilden Benutzeraktivitäten, für die kein Input-/Output-Mapping konfiguriert wurde und die nicht parallel zu anderen Aktivitäten ausgeführt werden. Diese Tokens besitzen denselben Geltungsbereich wie das direkt darüber liegende Token.

In der Ansicht des Tokens ohne eigenen Geltungsbereich finden Sie eine Verknüpfung zum Token aus dem darüber liegenden Bereich. Wenn Sie diese Verknüpfung auswählen, können Sie die Variablen des verknüpften Geltungsbereichs anzeigen.

### 1.4.6. Verwenden von Timer-Ereignissen

Bei der Prozessmodellierung können Sie das BPMN-Element **intermediate boundary event** (angeheftetes Zwischenereignis) vom Typ Timer verwenden. Diese Zwischenereignisse können Sie an Benutzeraktivitä-

ten oder Services heften. Zum Einstieg in das Arbeiten mit angehefteten Zwischenereignissen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen.

Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="timer_event_process" name="Timer Event Process"
isExecutable="true">
    <startEvent id="start" />
        <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

**Hinzufügen eines Zwischenereignisses**

Fügen Sie dem Prozess das BPMN-Element **boundaryEvent** hinzu.

```
...
<process id="timer_event_process" name="Timer Event Process"
isExecutable="true">
  ...
  <userTask ... />
  <boundaryEvent id="timer" attachedToRef="task_hello_world">
    <timerEventDefinition>
      <timeDuration>PT1M</timeDuration>
    </timerEventDefinition>
  </boundaryEvent>
  ...
</process>
```

**Erläuterung der Eigenschaften**

- **id:**Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **attachedToRef:** Die Eigenschaft definiert, an welches BPMN-Element dieses Zwischenereignis angeheftet werden soll.

**Erläuterung der Ereignisdefinition:**

- **timerEventDefinition:** Dieses BPMN-Element definiert, dass es sich bei dem Zwischenereignis um ein Zeitereignis handelt
- **timeDuration:** Dieses BPMN-Element definiert, dass es sich um eine relative Zeitspanne handelt. In diesem Fall ist eine Spanne von einer Minute definiert. Möchten Sie einen absoluten Zeitpunkt definieren, können Sie das BPMN-Element **timeDate** verwenden. In beiden Fällen müssen Sie die Werte im ISO-8601-Format angeben. Sie können die Werte auch aus Variablen ermitteln.

**Hinzufügen eines Ereignispfades**

Tritt im Prozess ein Zwischenereignis ein, folgt der Prozess einem separaten Prozesspfad. Sie können einen Prozess beispielsweise so konfigurieren, dass der Prozess von einem Zwischenereignis direkt zum Endereignis führt. Diese Konfiguration fügen Sie folgendermaßen in den Prozess ein:

```
...
<process id="timer_event_process" name="Timer Event Process"
```

```
isExecutable="true">
  ...
  <sequenceFlow id="s3" sourceRef="timer" targetRef="end" />
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/
20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
targetNamespace="" exporter="d.velop process modeler">
  <process id="timer_event_process" name="Timer Event Process"
isExecutable="true">
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <boundaryEvent id="timer" attachedToRef="task_hello_world">
      <timerEventDefinition>
        <timeDuration>PT1M</timeDuration>
      </timerEventDefinition>
    </boundaryEvent>
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <sequenceFlow id="s3" sourceRef="timer" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="timer_event_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="290" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="timer_di" bpmnElement="timer">
        <dc:Bounds x="372" y="139" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="462" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="117" />
        <di:waypoint x="290" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="390" y="117" />
        <di:waypoint x="462" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="408" y="157" />
```

```
        <di:waypoint x="480" y="157" />
        <di:waypoint x="480" y="135" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

## Verwenden von Timer-Startereignissen

In Ihrem Prozess können Sie das BPMN-Element **Timer Start Event** (Timer-Startereignis) verwenden, um Prozesse zeitgesteuert zu starten. Verwenden Sie das Element, wenn ein Prozess zu bestimmten Zeitpunkten oder in regelmäßigen Abständen automatisch starten soll.

### Hinzufügen eines Timer-Startereignisses

Um einen Prozess zeitgesteuert zu starten, können Sie in Ihrem Prozess ein Timer-Startereignis als Startpunkt verwenden. Erstellen Sie einen einfachen Prozess und erweitern Sie **startEvent** um **timerEventDefinition**. Legen Sie in **timerEventDefinition** mithilfe von **timeCycle** die genaue Zeitsteuerung fest. Die Information zur Zeitzone legen Sie über die Eigenschaft **timeZone** fest.

In folgendem Beispiel startet der Prozess jeden Tag um 12 Uhr:

```
...
<bpmn:process id="timer_start_prozess" name="Timer_Start_Prozess"
isExecutable="true">
...
<bpmn:startEvent id="start" name="Start">
    <bpmn:extensionElements>
        <camunda:properties>
            <camunda:property name="timeZone" value="Europe/Berlin"/>
        </camunda:properties>
    </bpmn:extensionElements>
    <bpmn:timerEventDefinition id="startTimer">
        <bpmn:timeCycle xsi:type="bpmn:tFormalExpression">0 0 12 1/1 * ? *</
bpmn:timeCycle>
    </bpmn:timerEventDefinition>
</bpmn:startEvent>
...
</bpmn:process>
...
```

### Erläuterung der Elemente

- **timerEventDuration**: Mit diesem BPMN-Element definieren Sie ein Timer-Startereignis, das nach einer festgelegten Dauer ausgelöst wird.
- **timeCycle**: In diesem BPMN-Element definieren Sie den Startzeitpunkt des Prozesses mit einem CRON-Ausdruck.
- **timeZone**: In diesem BPMN-Element definieren Sie die Zeitzone, auf die sich die festgelegte Startzeit bezieht.

### Konfigurieren des CRON-Ausdrucks

Ein CRON-Ausdruck besteht aus sieben Feldern, die den Zeitpunkt für den Start des Prozesses festlegen:

- Stelle 1: Sekunden
- Stelle 2: Minuten
- Stelle 3: Stunden

- Stelle 4: Tag im Monat
- Stelle 5: Monat
- Stelle 6: Tag der Woche
- Stelle 7: Jahr

Beispiel für einen CRON-Ausdruck:

```
0 0 12 1/1 * ? *
```

Erläuterung des Beispiels:

- 0 Sekunden
- 0 Minuten
- 12 Uhr
- Jeden Tag im Monat (1/1)
- Jeden Monat
- **?** im Feld **Tag der Woche**, da **Tag im Monat** verwendet wird
- Jedes Jahr

**Hinweise zu CRON-Ausdrücken**

- **Tag im Monat** vs. **Tag der Woche**: Die Felder **Tag im Monat** und **Tag der Woche** können Sie nicht gleichzeitig verwenden. Tragen Sie für eines der beiden Felder immer ein Fragezeichen ein.
- Minimales Intervall: Das kleinste mögliche Intervall beträgt eine Stunde. Eine minutengenaue Ausführung wird nicht sichergestellt.
- Zeitliche Begrenzung: Das Jahr der ersten Ausführung darf nicht mehr als 100 Jahre in der Zukunft liegen.

### 1.4.7. Verwenden von Verzweigungen

Beim Modellieren eigener Prozesse können Sie das BPMN-Element **Gateway** vom Typ exclusive (Verzweigung vom Typ exklusiv) verwenden, um dynamische Prozesse mit einer Entscheidungslogik zu erstellen.

Zum Einstieg in das Arbeiten mit exklusiven Verzweigungen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Dieser Prozess sieht beispielsweise so aus:


Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:camunda="http://
camunda.org/schema/1.0/bpmn" targetNamespace="" exporter="d.velop process
modeler">
  <process id="exclusive_gateway_process" name="Exclusive Gateway Process"
isExecutable="true">
    <startEvent id="start" />
        <userTask id="task_hello_world" name="Hello World"
camunda:camundaUsers="${variables.get('dv_initiator')}" />
```

```
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

Diesen Beispielprozess erweitern Sie nun mit einer Verzweigung. Anhand des Wertes einer Prozessvariablen führt dieser Prozess entweder zu der Benutzeraktivität oder wird direkt beendet.

**Hinzufügen einer Verzweigung**

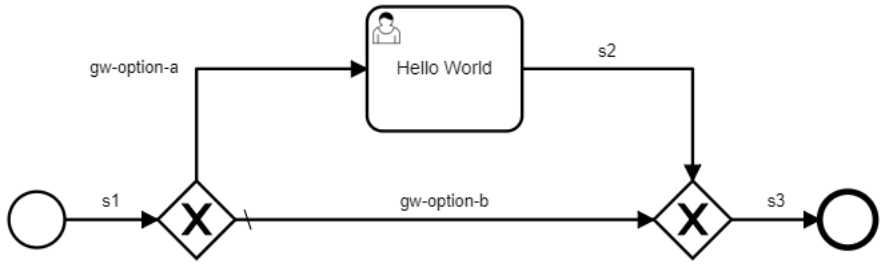
Fügen Sie dem Prozess zunächst das BPMN-Element **exclusiveGateway** hinzu. Das **exclusiveGateway**-Element repräsentiert eine Verzweigung mit nur einem einzigen möglichen Ausgangspfad. Ersetzen Sie die vorhandenen Sequenzflüsse anschließend mit den folgenden Sequenzflüssen:

```
...
<process id="exclusive_gateway_process" name="Exclusive Gateway Process"
isExecutable="true">
  <startEvent id="start" />
  <exclusiveGateway id="gateway" default="gateway-option-b" />
  <userTask id="task_hello_world" name="Hello World" camunda:camundaUsers="$
{variables.get('dv_initiator')}" />
  <exclusiveGateway id="join" />
  <endEvent id="end" />
  <sequenceFlow id="s1" sourceRef="start" targetRef="gateway" />
  <sequenceFlow id="gateway-option-a" sourceRef="gateway"
targetRef="task_hello_world" />
  <sequenceFlow id="gateway-option-b" sourceRef="gateway" targetRef="join"
/>
  <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="join" />
  <sequenceFlow id="s3" sourceRef="join" targetRef="end" />
</process>
```

**Erläuterung der Eigenschaften:**

- **id:**Die Eigenschaft wird als eindeutiger Bezeichner für die Verzweigung verwendet.
- **default:** Die Eigenschaft definiert die ID des Sequenzflusses, der als Standard-Ausgangspfad der Verzweigung gewählt werden soll.

Der Prozess sieht dann beispielsweise so aus:




**Hinzufügen einer Bedingung**

Alle Ausgangspfade einer exklusiven Verzweigung müssen eine Bedingung enthalten. Mithilfe dieser Bedingung ermittelt die Anwendung, welchem Pfad der Prozess folgen soll. Für den Beispielprozess fügen Sie dem Sequenzfluss **gw-option-a** die Bedingung hinzu, dass dieser Pfad abhängig von der Variable **amount** gewählt wird. Entspricht diese Variable einem Wert größer 1000, soll der Sequenzfluss **gw-option-a** gewählt werden.

```
...
<sequenceFlow id="gw-option-a" sourceRef="gateway"
targetRef="task_hello_world" >
  <conditionExpression
xsi:type="tFormalExpression">#{variables.get("amount") > 1000}</
conditionExpression>
<sequenceFlow>
...
```

Um eine Prozessvariable verwenden zu können, müssen Sie diese der Prozessdefinition hinzufügen. In der Prozessdefinition fügen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** zum BPMN-Element **process** hinzu. Zum Definieren der Prozessvariable fügen Sie dem BPMN-Element **camunda:properties** noch das Element **camunda:property** hinzu. In diesem Beispiel fügen Sie die Variable **amount** vom Typ **Number** ein. Diese ist eine Pflichtvariable, die beim Start mitgegeben werden muss.

```
...
<process id="exclusive_gateway_process" name="Exclusive Gateway Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="variable:amount*" value="Number!" />
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:camunda="http://
camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://www.omg.org/spec/BPMN/
20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI" targetNamespace=""
exporter="d.velop process modeler">
  <process id="exclusive_gateway_process" name="Exclusive Gateway Process"
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:amount*" value="Number!" />
      </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <exclusiveGateway id="gateway" default="gateway-option-b" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <exclusiveGateway id="join" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="gateway" />
    <sequenceFlow id="gateway-option-a" sourceRef="gateway"
targetRef="task_hello_world">
      <conditionExpression
xsi:type="tFormalExpression">#{variables.get("amount") > 1000}</
conditionExpression>
    </sequenceFlow>
    <sequenceFlow id="gateway-option-b" sourceRef="gateway"
```

```
targetRef="join" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="join" />
    <sequenceFlow id="s3" sourceRef="join" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1"
bpmnElement="exclusive_gateway_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="199" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="gateway_di" bpmnElement="gateway"
isMarkerVisible="true">
        <dc:Bounds x="275" y="192" width="50" height="50" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="410" y="80" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="join_di" bpmnElement="join"
isMarkerVisible="true">
        <dc:Bounds x="595" y="192" width="50" height="50" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="702" y="199" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="217" />
        <di:waypoint x="275" y="217" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="gateway-option-a_di" bpmnElement="gateway-option-
a">
        <di:waypoint x="300" y="192" />
        <di:waypoint x="300" y="120" />
        <di:waypoint x="410" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="gateway-option-b_di" bpmnElement="gateway-option-
b">
        <di:waypoint x="325" y="217" />
        <di:waypoint x="595" y="217" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="510" y="120" />
        <di:waypoint x="620" y="120" />
        <di:waypoint x="620" y="192" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="645" y="217" />
        <di:waypoint x="702" y="217" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

Sie können den Wert der Variablen **amount** nun zum Beispiel beim Start des Prozesses belegen, und so den Prozessverlauf beeinflussen.

### 1.4.8. Einfügen von Benutzeraktivitäten

In diesem Thema erhalten Sie grundlegende Informationen über die Verwendung und Anpassung des BPMN-Elements **User Task** (Benutzeraktivität).

### Wissenswertes zu Benutzeraktivitäten

Im BPMN-Standard gibt es unterschiedliche Konstrukte für die Zuweisung von Empfängern innerhalb eines **User Task** (Benutzeraktivität).

Wir empfehlen die Verwendung der Eigenschaft **camunda:candidateUsers**.

#### Potential Owner

```
<userTask id='theTask' name='important task' camunda:candidateUsers="${variables.get('myPerformer')}" />
```

#### Verwenden von Konstanten für Empfänger

Wenn Sie die Empfänger einer Aktivität als Konstante angeben möchten, verwenden Sie folgende Syntax:

```
${collections.from("identity:///identityprovider/scim/users/someUserId")}
// or
${collections.from("identity:///identityprovider/scim/users/someUserId",
"identity:///identityprovider/scim/users/someOtherUserId")}
```

#### Verwenden von Variablen für Empfänger

Wenn die Empfänger einer Aktivität aus Variablen ermittelt werden sollen, verwenden Sie folgende Syntax für die Referenzierung der Variablen:

```
${variables.get("variableName")}
```

#### Verlinken von Zusatzinformationen zu Benutzeraktivitäten

Um Benutzeraktivitäten Zusatzinformationen für die Bearbeitung hinzuzufügen, können Sie die dafür reservierte Prozessvariable **dv_attachment** verwenden.

In dieser Variable können Sie eine URL speichern. Die URL wird bei Erstellung einer Aktivität hinzugefügt, wenn Sie sie zuvor eingetragen haben.

> **Warnung**
> Die Prozessvariable **dv_attachment** ist mit dem Datentyp **DmsObject** vordefiniert. Möchten Sie eine andere URI als die zu einem Objekt der DMS-App verwenden, müssen Sie den Wert des Datentyps auf **String** ändern.

#### Hinzufügen von Metadaten zu Benutzeraktivitäten

Um Metadaten zu Benutzeraktivitäten hinzuzufügen, können Sie die Metadaten innerhalb des BPMN-Elements **userTask** definieren. Fügen Sie das Element **extensionElements** und darunter das Element **camunda:properties** hinzu, falls die Prozessdefinition diese Elemente noch nicht enthält. Unterhalb des Elements **camunda:properties** können Sie nun wie folgt Metadaten definieren:

```
<userTask>
...
        <extensionElements>
```

```
                <camunda:properties>
                        <camunda:property
name="metadata:invoiceNumber:value" value="$
{variables.get("invoiceNumber")}" />
                <camunda:property name="metadata:invoiceNumber:caption"
value="Invoice Number" />
                <camunda:property name="metadata:invoiceNumber:caption:de"
value="Rechnungsnummer" />
                <camunda:property name="metadata:invoiceNumber:type"
value="String" />
        </camunda:properties>
        </extensionElements>
...
</userTask>
```

Das Attribut **name** innerhalb des Elements **camunda:property** kann folgende Werte annehmen:

| Wert | Bedeutung | |
|---|---|---|
| metadata:<key>:value | Definiert den Wert eines Metadatums. | Der Key des Metadatums muss folgendem Ausdruck entsprechen: [A-Za-z0-9]{1,255} |
| metadata:<key>:caption | (Optional) Definiert die Standardbeschriftung eines Metadatums.<br><br>Falls nicht vorhanden, wird der Schlüssel als Beschriftung verwendet. | |
| metadata:<key>:caption:<language> | (Optional) Definiert die Beschriftung eines Metadatums für die angegebene Sprache. Das Sprachkürzel muss der ISO-639-1 Norm entsprechend aus zwei Buchstaben bestehen.<br><br>Falls nicht vorhanden, wird die Standardbeschriftung bzw. der Schlüssel verwendet. | |
| metadata:<key>:type | (Optional) Definiert den Datentyp des Metadatums. | Entspricht dem Datentyp aus der TaskApp (**String**, **Number**, **Money**, **Date**). Wenn kein Typ angegeben wird, wird der Typ **String** angenommen. |

Das Attribut **value** kann entweder einen festen Wert oder einen Ausdruck enthalten. Dieser Wert darf maximal 255 Zeichen lang sein. Bei der Verwendung eines Ausdrucks gilt diese Beschränkung für das Ergebnis des Ausdrucks. Darüber hinaus muss das Ergebnis eines Ausdrucks ein einzelner Wert sein. Werte vom Typ **Collection** oder **Array** sind nicht erlaubt.

**Bestimmen der Aufbewahrungsdauer einer Benutzeraktivität**

Sie können die Aufbewahrungsdauer von Benutzeraktivitäten innerhalb des BPMN-Elements **userTask** definieren. Fügen Sie das Element **extensionElements** und darunter das Element **camunda:properties** hinzu, falls die Prozessdefinition diese Elemente noch nicht enthält. Unterhalb des Elements **camunda:properties** können Sie nun wie folgt die Aufbewahrungsdauer definieren:

```
<userTask>
...
        <extensionElements>
                <camunda:properties>
                        <camunda:property name="retentionTime" value="P7D"
/>
                </camunda:properties>
        </extensionElements>
...
</userTask>
```

Das Attribut **value** kann entweder einen festen Wert oder einen Ausdruck enthalten. Definieren Sie die Aufbewahrungsdauer als Zeitspanne nach ISO-8601 in Tagen, z.B. **P30D** für 30 Tage.

#### Bestimmen der Nutzbarkeit von Aktionen in der Benutzeroberfläche einer Benutzeraktivität

Sie können die Nutbarkeit von Aktionen in der Benutzeroberfläche der Benutzeraktivität innerhalb des BPMN-Elements **userTask** definieren. Fügen Sie das Element **extensionElements** und darunter das Element **camunda:properties** hinzu, falls die Prozessdefinition diese Elemente noch nicht enthält. Unterhalb des Elements **camunda:properties** können Sie nun wie folgt die Aufbewahrungsdauer definieren:

```
<userTask>
...
        <extensionElements>
                <camunda:properties>
                        <camunda:property name="actionScope:complete"
value="[list, details]" />
                </camunda:properties>
        </extensionElements>
...
</userTask>
```

Die Eigenschaft **name** des Elements enthält das Prefix **actionScope** gefolgt von der zu konfigurierenden Aktion (in diesem Beispiel **complete**). Die Eigenschaft **value** enthält die gewünschten Optionen für die Nutzbarkeit. Welche Aktionen und Nutzbarkeitsoptionen möglich sind, können Sie der API-Dokumentation der TaskApp entnehmen.

## Hinzufügen von Benutzeroberflächen zu einer Benutzeraktivität

Wenn Sie bei der Erstellung eines Prozessmodells das BPMN-Element **User Task** (Benutzeraktivität) verwenden, können Sie der Aktivität eine Benutzeroberfläche hinzufügen. Dem Benutzer wird dann beispielsweise eine Schaltfläche angezeigt, mit der er eine Aufgabe beginnen kann.

Um eine Benutzeroberfläche hinzuzufügen, verwenden Sie die Eigenschaft **f ormKey**.

#### Beispiel

#### Form Key

```
<userTask id="someTask" camunda:formKey="uri:/myapp/myform">
  ...
</userTask>
```

Der Schlüssel besteht aus einem Präfix und einem Wert. Diese werden durch einen Doppelpunkt voneinander getrennt. Folgende Präfixe können verwendet werden:

| Präfix | Bedeutung | Beispiel |
|---|---|---|
| uri | Die Benutzeroberfläche wird über eine URI angegeben. | uri:https://example.com/ |

#### Verwenden von Prozessvariablen

Der Schlüssel kann auch Prozessvariablen beinhalten. Diese können in Form von Expression Language angegeben werden.

#### Beispiel

```
uri:${variables.get("formUri")}

uri:https://example.com/?queryParam=${variables.get("formParam")}
```

#### Externer Zugriff auf Prozessvariablen

Mit dem Ausdruck **${process.task.variablesUri}** kann ein URI erzeugt werden, um einer externen Applikation den Zugriff auf die aktuellen Variablen dieser Benutzeroberfläche zu ermöglichen. Mittels der HTTP-Methoden **GET** und **PUT** können diese Variablen gelesen und verändert werden. In dieser Bearbeitungsoberfläche muss zu den verwendenden Variablen ein Input- und Output-Mapping definiert sein.

**Beispiel**

```
uri:/myApp?variablesUri=${process.task.variablesUri}
```

## Verwenden von Prozessvariablen in einer Benutzeroberfläche

Enthält eine Benutzeraktivität ein Formular, können Sie mit HTTP lesend und schreibend auf die verwendeten Prozessvariablen zugreifen.

Zum Einstieg in das Arbeiten mit Prozessvariablen in einem Formular ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Dieser Prozess sieht beispielsweise so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="GettingStarted" name="Example: Getting started"
isExecutable="true">
    <startEvent id="start"/>
    <userTask id='task_hello_world' name='Hello World'
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end"/>
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

Zur Vereinfachung sind die grafischen Informationen zum BPMN-Diagramm in dieser BPMN-Beispieldatei nicht enthalten.

**Erstellen eines Formulars**

Erstellen Sie zunächst eine HTML-Datei. Diese kann wie folgt aussehen. Die Code-Beispiele sind auf die Funktionsweise in Google Chrome ausgerichtet. Andere Browser können möglicherweise eine andere Syntax erfordern.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Getting started</title>
</head>
<body/>
</html>
```

Legen Sie die HTML-Datei so ab, dass sie mit einem Http-Gateway erreichbar ist. Fügen Sie den URI des HTML-Dokuments zum BPMN-Element **userTask** hinzu.

```
...
<process id="GettingStarted" name="Example: Getting started"
isExecutable="true">
        ...
    <userTask id='task_hello_world'
name='Hello World' camunda:formKey="uri:/res/process/process-
```

```
form.html?variables=${process.task.variablesUri}" camunda:candidateUsers="$
{variables.get('dv_initiator')}" />
    ...
```

**Erläuterung der Eigenschaft**

- **camunda:formKey**: In dieser Eigenschaft wird der URI definiert, der die Oberfläche für die Benutzerakt-ivität bereitstellt. Das Präfix **uri:** ist dabei verpflichtend. Die Variable **${process.task.variablesUri}** wird als Query-Parameter zum URI ergänzt. Die Variable wird von der Anwendung zur Laufzeit in einen URI aufgelöst, der einen HTTP-Endpunkt zum Lesen und Schreiben der Variablen bereitstellt.

**Erstellen der Benutzeroberfläche**

Erstellen Sie zunächst das Grundgerüst für eine HTML-Tabelle in Ihrem HTML-Dokument. Im späteren Verlauf werden die Variablen dann in dieser Tabelle angezeigt.

```
...
<body>
    <table id="variables">
                <thead>
                        <tr>
                                <th>Name</th>
                                <th>Value</th>
                        </tr>
                </thead>
                <tbody id="variableValues">
                </tbody>
        </table>
</body>
...
```

**Abfragen der Prozessvariablen**

Im Head des HTML-Dokuments erstellen Sie eine Funktion zum Auslesen der Prozessvariablen. Die Prozessvariablen werden mit einer HTTP-GET-Anforderung entsprechend der Rest-API von der Anwendung abgefragt. Das Laden der Variablen wird durch das **onLoad**-Event des **body**-Elements ausgelöst.

```
...
<head>
        ...
        <script type="text/javascript">

                var urlParams = new URLSearchParams(window.location.search);
                var variablesUri = urlParams.get('variables');

                function loadVariables() {
                        fetch(variablesUri)
                                .then(response => response.json());
                }

        </script>
</head>
<body onload="loadVariables()">
...
```

Nachdem die Variablen geladen wurden, können Sie diese nun in der vorbereiteten Tabelle anzeigen. Im folgenden Beispiel wird die JavaScript-Bibliothek jQuery verwendet. Fügen Sie im Head des HTML-

Dokuments die jQuery-Bibliothek hinzu und implementieren Sie entsprechende Methoden zum Anzeigen der Variablen.

```
...
<head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/
jquery.min.js"></script>
        <script type="text/javascript">

                var urlParams = new URLSearchParams(window.location.search);
                var variablesUri = urlParams.get('variables');

                function loadVariables() {
                        fetch(variablesUri)
                                .then(response => response.json())
                                .then(json =>
createTableFromVariables(json.variables || {}));
                }

                function createTableFromVariables(variables) {
                        Object.keys(variables).forEach(variable => {
                                var value = variables[variable];
                                $('#variableValues').append(
`<tr class="variable">${createKeyColumn(variable)}$
{createValueColumn(value)}</tr>`);
                        });
                }

                function createKeyColumn(variable) {
                        return `<td><span class="key">${variable}</span></
td>`;
                }

                function createValueColumn(value) {
                        return `<td><input class="value" type="text"
value="${value}"></td>`;
                }

        </script>
</head>
<body onload="loadVariables()">
...
```

Die Variablen werden nun nach dem Laden des HTML-Dokuments abgefragt und angezeigt.

**Schreiben der Prozessvariablen**

Fügen Sie in den Skript-Bereich Methoden ein, die das Zurückschreiben der Variablen in die Anwendung ermöglichen. Anschließend fügen Sie einen **button** unterhalb der Tabelle hinzu, der die Methode **saveVariables** aufruft.

```
...
<head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/
jquery.min.js"></script>
        <script type="text/javascript">
```

```
                var urlParams = new URLSearchParams(window.location.search);
                var variablesUri = urlParams.get('variables');

                ...

                function saveVariables() {
                        var variables = createVariablesFromTable();

                        // These variables are read-only and must not be
sent to server when setting variables
                        delete variables['dv_initiator'];
                        delete variables['dv_start_time'];

                    const headers = new Headers();
                        headers.append('Cache-Control', 'no-cache');
                        headers.append('Content-Type', 'application/json');

                        let promise = fetch(variablesUri, {
                                method: 'PUT',
                                headers: headers,
                                credentials: 'same-origin',
                                body: JSON.stringify({variables})
                        });

                        promise.then(function(response) {
                                let status = response.status;
                                if (status !== 200 ) {
                                        window.alert("Saving variables
failed: " + status);
                                } else {
                                        window.alert("Variables saved");
                                }

                        });
                }

                function createVariablesFromTable() {
                        var variables = {};
                        $("tr.variable").each(function() {
                                $this = $(this);
                                var key = $this.find("span.key").html();
                                var value = $this.find("input.value").val();
                                variables[key] = value;
                        });
                        return variables;
                }

        </script>
</head>
<body onload="loadVariables()">
        ...
        <br/>
        <button onClick="saveVariables()">Save</button>
</body>
```

Mithilfe des Formulars, das bei der Ausführung des Prozesses angezeigt wird, können Sie die aktuellen Prozessvariablen anzeigen und ändern.

Das finale HTML-Dokument sieht wie folgt aus. Das HTML-Dokument enthält einige ergänzende CSS-Definitionen zur optimierten Darstellung der Oberflächenelemente.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Process Form</title>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/
jquery.min.js"></script>
        <style>
                table {
                  border-collapse: collapse;
                  width: 100%;
                }

                table, th, td {
                  border: 1px solid black;
                }

                td {
                  padding: 3px;
                }

                .key {
                        color: gray;
                }

                th {
                  text-align: left;
                }

                input {
                        width: 100%;
                        border: 0;
                }
        </style>
    <script type="text/javascript">

                var urlParams = new URLSearchParams(window.location.search);
                var variablesUri = urlParams.get('variables');

                function loadVariables() {
                        fetch(variablesUri)
                                .then(response => response.json())
                                .then(json =>
createTableFromVariables(json.variables || {}));
                }

                function saveVariables() {
                        var variables = createVariablesFromTable();

                        // These variables are read-only and must not be
```

```
sent to server when setting variables
                        delete variables['dv_initiator'];
                        delete variables['dv_start_time'];

                const headers = new Headers();
                        headers.append('Cache-Control', 'no-cache');
                        headers.append('Content-Type', 'application/json');

                        let promise = fetch(variablesUri, {
                                method: 'PUT',
                                headers: headers,
                                credentials: 'same-origin',
                                body: JSON.stringify({variables})
                        });

                        promise.then(function(response) {
                                let status = response.status;
                                if (status !== 200 ) {
                                        window.alert("Saving variables
failed: " + status);
                                } else {
                                        window.alert("Variables saved");
                                }

                        });
                }

                function createTableFromVariables(variables) {
                        Object.keys(variables).forEach(variable => {
                                var value = variables[variable];
                                $('#variableValues').append(
`<tr class="variable">${createKeyColumn(variable)}$
{createValueColumn(value)}</tr>`);
                        });
                }

                function createKeyColumn(variable) {
                        return `<td><span class="key">${variable}</span></
td>`;
                }

                function createValueColumn(value) {
                        return `<td><input class="value" type="text"
value="${value}"></td>`;
                }

                function createVariablesFromTable() {
                        var variables = {};
                        $("tr.variable").each(function() {
                                $this = $(this);
                                var key = $this.find("span.key").html();
                                var value = $this.find("input.value").val();
                                variables[key] = value;
                        });
                        return variables;
```

```
                }

        </script>
</head>
<body onload="loadVariables()">
    <table id="variables">
                <thead>
                        <tr>
                                <th>Name</th>
                                <th>Value</th>
                        </tr>
                </thead>
                <tbody id="variableValues">
                </tbody>
        </table>
        <br/>
        <button onClick="saveVariables()">Save</button>
</body>
</html>
```

### 1.4.9. Verwenden einer Multi-Instanz

Sie können Mehrfachausführungen zu Benutzeraktivitäten und Subprozessen hinzufügen. Voraussetzung dafür ist, dass die Aktivität oder der Prozess auf einer Variable mit Mehrfachwert basiert. Für jeden Wert der Variable wird eine Ausführung der Aktivität vorgenommen.


Zum Einstieg in das Arbeiten mit Multi-Instanzen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Dieser Prozess sieht beispielsweise so aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="multi_instance_process" name="Multi Instance Process"
isExecutable="true">
    <startEvent id="start" />
        <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

#### Hinzufügen einer Multi-Instanz

Fügen Sie der Benutzeraktivität das BPMN-Element **multiInstanceLoopCharacteristics** hinzu. Ändern Sie anschließend die Variable, die den Empfänger der Benutzeraktivität bestimmt, in **myPerformer**.

```
...
<process id="multi_instance_process" name="Multi Instance Process"
isExecutable="true">
```

```
  ...
  <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('myPerformer')}" >
      <multiInstanceLoopCharacteristics
          isSequential="false"
          camunda:collection="${variables.get('performers')}"
          camunda:elementVariable="myPerformer" />
  </userTask>
  ...
</process>
```

**Erläuterung der Eigenschaften**

- **isSequential**: Verwenden Sie den Wert **true**, um die Aktivitäten nacheinander auszuführen und **false**, um sie gleichzeitig auszuführen.
- **camunda:collection**: Ausdruck, der eine Mehrfachvariable zurückgibt.
- **camunda:elementVariable**: Variablenname, der lokal für den jeweiligen Wert verwendet wird. Für die Variable muss eine Definition vorhanden sein.

Im Beispielprozess wird für jeden Wert der Variable **performers** eine Aktivität zugestellt. Mit der lokalen Variable **myPerformer** kann jedes einzelne Element als Empfänger angegeben werden.

Damit die Variablen **performers** und **myPerformer** verwendet werden können, müssen Sie diese definieren. Fügen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** zum BPMN-Element **process** hinzu. Zum Definieren der Prozessvariablen fügen Sie den BPMN-Elementen **camunda:properties** noch jeweils das Element **camunda:property** hinzu.

```
...
<process id="multi_instance_process" name="Multi Instance Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="variable:performers*" value="[Identity]!" />
      <camunda:property name="variable:myPerformer" value="Identity"
/>
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/
20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
targetNamespace="" exporter="d.velop process modeler">
  <process id="multi_instance_process" name="Multi Instance Process"
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:performers*" value="[Identity]!" />
        <camunda:property name="variable:myPerformer" value="Identity"
/>
      </camunda:properties>
```

```
    </extensionElements>
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('myPerformer')}" >
      <multiInstanceLoopCharacteristics
            isSequential="false"
            camunda:collection="${variables.get('performers')}"
            camunda:elementVariable="myPerformer" />
    </userTask>
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="multi_instance_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="290" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="462" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="117" />
        <di:waypoint x="290" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="390" y="117" />
        <di:waypoint x="462" y="117" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

Den Wert der Variablen **performers** müssen Sie nun beim Start des Prozesses belegen.

### 1.4.10. Verwenden eines Startformulars

Sie können ein Formular zu einem Startereignis hinzufügen, wie beispielsweise einen Urlaubsantrag. Sie können das Startformular dann als Einstiegspunkt für diesen Prozess verwenden. Um einem Startereignis ein Formular hinzuzufügen, verwenden Sie die Eigenschaft **formKey**.

```
<startEvent id="start" name="start" camunda:formKey="uri:/myapp/mystart">
    ...
</startEvent>
```

### 1.4.11. Verwenden von Services

In diesem Thema erfahren Sie, wie Sie Services in Ihre Prozesse einbinden können. Beim Einfügen von Services können Sie sich entscheiden, ob der Service synchron oder asynchron durchgeführt wird.

## Verwenden eines synchronen Service

Automatisierte Aktivitäten werden in einem Prozess mithilfe von Services ausgeführt. Für die Verwendung von synchronen Services in BPMN können Sie das BPMN-Element **Send Task** (Sendenaktiviät) verwenden. Erreicht der Prozess die Sendenaktivität, so werden die im Prozess definierten Daten an den HTTP-Endpunkt des Serviceanbieters geschickt und unmittelbar auf das Ergebnis gewartet. Anschließend wird die Prozessausführung direkt fortgesetzt. Dieser Artikel zeigt Ihnen anhand eines einfachen Beispielservice, wie Sie einen synchronen Service verwenden können. Der Beispielservice "Hello World" wandelt einen beliebigen Text in "Hello <yourname>" um, wobei <yourname> der Text ist, der zuvor an den Service gesendet wurde.

Um dieses Beispiel durchzuführen, ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

**Hinzufügen einer Sendenaktivität**

Fügen Sie zunächst eine Sendenaktivität nach dem Startereignis hinzu.

```
...
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  <startEvent id="start" />
  <sendTask id="call_service"
        name="Call Service 'Hello World'"
        camunda:asyncBefore="true"
            camunda:asyncAfter="true"
        camunda:delegateExpression="${syncService}"
        camunda:exclusive="true">
  </sendTask>
  ...
</process>
```

**Erläuterung der Eigenschaften:**

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **camunda:***: Diese Eigenschaften beinhalten technische Informationen, die zur Ausführung benötigt werden.

> **Anmerkung**
> Bei **synchronen Services** müssen Sie für die **camunda**-Eigenschaften immer die Werte des Beispiels eintragen. Wenn Sie abweichende Werte eintragen, kann der Prozess nicht bereitgestellt werden.

Fügen Sie nun der Sendenaktivität die BPMN-Elemente **extensionElements** und **camunda:inputOutput** hinzu. Sie müssen alle Daten definieren, die an den HTTP-Endpunkt des Services gesendet werden sollen. Zur Definition der Daten fügen Sie für jeden Wert das BPMN-Element **camunda:inputParameter** in die Prozessdefinition ein. Anschließend müssen Sie auch die erwarteten Ausgabewerte definieren. Fügen Sie hierzu für jeden Wert das BPMN-Element **camunda:outputParameter** in die Prozessdefinition ein.

```
...
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  <startEvent id="start" />
  <sendTask id="call_service" ...>
    <extensionElements>
      <camunda:inputOutput>
        <camunda:inputParameter name="service.uri">/process/services/
helloworld/sync</camunda:inputParameter>
        <camunda:inputParameter name="yourName">$
{variables.getDisplayValue("dv_initiator")}</camunda:inputParameter>
        <camunda:outputParameter name="greeting">$
{variables.get("greeting")}</camunda:outputParameter>
      </camunda:inputOutput>
    </extensionElements>
  </sendTask>
  ...
</process>
```

**Erläuterung der Parameter:**

- **service.uri**: Dieser Parameter muss immer eingefügt werden. Er definiert den URI, unter dem der HTTP-Endpunkt des Services erreichbar ist. Das Beispiel zeigt den URI des "Hello World"-Services. Der Wert dieser Variablen muss eine Konstante sein, das heißt er darf keine Ausdrücke enthalten.
- **yourName**: Dieser Parameter wurde vom konkreten Service als Eingabewert definiert. In diesem Beispiel wird der Anzeigename des Benutzers verwendet, der den Prozess gestartet hat.
- **greeting**: Der Service hat als einzigen Ausgabewert den Parameter **greeting** definiert. Dieser Parameter wird bei der Antwort des Services automatisch in eine gleichnamige Variable im Geltungsbereich der Sendenaktivität geschrieben. Der Parameter definiert, dass der Inhalt dieses Ergebniswertes bei Fortführung des Prozesses in die Variable **greeting** im Geltungsbereich des gesamten Prozesses geschrieben wird.

Um die Kommunikation zwischen der Anwendung und dem Service zu ermöglichen, müssen Sie die Schnittstelle des Services zum Prozess hinzufügen. Zum Hinzufügen der Schnittstelle tragen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** in die Prozessdefinition ein. Anschließend fügen Sie je ein BPMN-Element **camunda:property** für den Service-Eingabewert **yourName**, den Service-Ausgabewert **greeting**, sowie die resultierende Prozessvariable **greeting** ein. Alle Werte sind vom Datentyp **String**.

```
...
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  <extensionElements>
```

```
    <camunda:properties>
      <camunda:property name="service:/process/services/helloworld/
sync:in:yourName" value="String" />
      <camunda:property name="service:/process/services/helloworld/
sync:out:greeting" value="String" />
         <camunda:property name="service:/process/services/helloworld/
sync:name" value="Hello World" />
      <camunda:property name="variable:greeting" value="String" />
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Die Variable **greeting** enthält nun den Text "Hello <display name of process start user>". Verwenden Sie diese Variable nun als Namen für die Benutzeraktivität, damit der Anwender den freundlichen Gruß als Betreff der zu erledigenden Aufgabe erhält.

```
...
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  ...
  <userTask id="task_hello_world" name="${variables.get('greeting')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
  ...
</process>
```

Damit die Prozessaktivitäten in der richtigen Reihenfolge ausgeführt werden, müssen Sie die Sequenzflusselemente noch an die neuen Aktivitäten anpassen.

```
<process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
  ...
  <sequenceFlow id="s1" sourceRef="start" targetRef="call_service" />
  <sequenceFlow id="s2" sourceRef="call_service"
targetRef="task_hello_world" />
  <sequenceFlow id="s3" sourceRef="task_hello_world" targetRef="end" />
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI" xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
  <process id="sync_service_process" name="Sync Service Process"
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="service:/process/services/helloworld/
sync:in:yourName" value="String" />
        <camunda:property name="service:/process/services/helloworld/
sync:out:greeting" value="String" />
        <camunda:property name="service:/process/services/helloworld/
sync:name" value="Hello World" />
```

```
        <camunda:property name="variable:greeting" value="String" />
      </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <sendTask id="call_service"
          name="Call Service 'Hello World'"
          camunda:asyncBefore="true"
          camunda:asyncAfter="true"
          camunda:delegateExpression="${syncService}"
          camunda:exclusive="true">
      <extensionElements>
        <camunda:inputOutput>
          <camunda:inputParameter name="service.uri">/process/services/
helloworld/sync</camunda:inputParameter>
          <camunda:inputParameter name="yourName">$
{variables.getDisplayValue("dv_initiator")}</camunda:inputParameter>
          <camunda:outputParameter name="greeting">$
{variables.get("greeting")}</camunda:outputParameter>
        </camunda:inputOutput>
      </extensionElements>
    </sendTask>
    <userTask id="task_hello_world" name="${variables.get('greeting')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="call_service" />
    <sequenceFlow id="s2" sourceRef="call_service"
targetRef="task_hello_world" />
    <sequenceFlow id="s3" sourceRef="task_hello_world" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane" bpmnElement="sync_service_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="173" y="102" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="209" y="120" />
        <di:waypoint x="250" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="call_service_di" bpmnElement="call_service">
        <dc:Bounds x="250" y="80" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="350" y="120" />
        <di:waypoint x="390" y="120" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="390" y="80" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="490" y="120" />
        <di:waypoint x="532" y="120" />
      </bpmndi:BPMNEdge>
```

```
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="532" y="102" width="36" height="36" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

## Verwenden eines asynchronen Service

Automatisierte Aktivitäten werden in einem Prozess mithilfe von Services ausgeführt. Für die Verwendung von asynchronen Services in BPMN können Sie die BPMN-Elemente **Send Task** (Sendenaktivität) und **Receive Task** (Empfangenaktiviät) verwenden. Erreicht der Prozess die Sendenaktivität, so werden die im Prozess definierten Daten an den HTTP-Endpunkt des Serviceanbieters geschickt. Anschließend wird die Prozessausführung so lange pausiert, bis der Serviceanbieter das Ergebnis der Verarbeitung zurückmeldet. Dieser Artikel zeigt Ihnen anhand eines einfachen Beispielservice, wie Sie einen asynchronen Service verwenden können. Der Beispielservice "Hello World" wandelt einen beliebigen Text in "Hello <yourname>" um, wobei <yourname> der Text ist, der zuvor an den Service gesendet wurde.

Um dieses Beispiel durchzuführen, ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität erstellen. Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="async_service_process" name="Async Service Process"
isExecutable="true">
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
  </process>
</definitions>
```

### Hinzufügen einer Sendenaktivität

Fügen Sie zunächst eine Sendenaktivität nach dem Startereignis hinzu.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <startEvent id="start"/>
  <sendTask id="call_service"
        name="Call Service 'Hello World'"
        camunda:asyncBefore="true"
        camunda:delegateExpression="${asyncService}"
        camunda:exclusive="true">
  </sendTask>
  ...
</process>
```

**Erläuterung der Eigenschaften:**

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.

- **camunda:***: Diese Eigenschaften beinhalten technische Informationen, die zur Ausführung benötigt werden.

> **Anmerkung**
> Bei **asynchronen Services** müssen Sie für die **camunda**-Eigenschaften immer die Werte des Beispiels eintragen. Wenn Sie abweichende Werte eintragen, kann der Prozess nicht bereitgestellt werden.

Fügen Sie nun der Sendenaktivität die BPMN-Elemente **extensionElements** und **camunda:inputOutput** hinzu. Sie müssen alle Daten definieren, die an den HTTP-Endpunkt des Services gesendet werden sollen. Zur Definition der Daten fügen Sie für jeden Wert das BPMN-Element **camunda:inputParameter** in die Prozessdefinition ein.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <startEvent id="start"/>
  <sendTask id="call_service" ...>
    <extensionElements>
      <camunda:inputOutput>
        <camunda:inputParameter name="service.uri">/process/services/
helloworld/async</camunda:inputParameter>
        <camunda:inputParameter name="yourName">$
{variables.getDisplayValue("dv_initiator")}</camunda:inputParameter>
                <camunda:inputParameter name="service.callbackBase">/</
camunda:inputParameter>
          </camunda:inputOutput>
    </extensionElements>
  </sendTask>
  ...
</process>
```

**Erläuterung der Parameter:**

- **service.uri**: Dieser Parameter ist obligatorisch und muss daher immer eingefügt werden. Der Parameter definiert den URI, unter dem der HTTP-Endpunkt des Services erreichbar ist. Das Beispiel zeigt den URI des "Hello World"-Services. Der Wert dieser Variablen muss eine Konstante sein, das heißt, der Wert darf keine Ausdrücke enthalten.
- **yourName**: Dieser Parameter wird vom konkreten Service als Eingabewert definiert. In diesem Beispiel wird der Anzeigename des Benutzers verwendet, der den Prozess gestartet hat.

**Änderung der Antwort-URIs**

Sie können die Origin der URIs ändern, die dem Service für die Antwort übergeben werden. Hierfür stehen Ihnen die folgenden zwei Optionen zur Verfügung.

**Angeben der Origin am SendTask eines konkreten Services**

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <startEvent id="start"/>
  <sendTask id="call_service" ...>
    <extensionElements>
      <camunda:inputOutput>
        ...
```

```
                <camunda:inputParameter name="service.callbackBase">/</
camunda:inputParameter>
                ...
            </camunda:inputOutput>
    </extensionElements>
  </sendTask>
  ...
</process>
```

Fügen Sie dem SendTask einen weiteren **camunda:inputParameter** mit dem Namen **service.callbackBase** hinzu. Dieser Parameter gilt dann für diesen konkreten Serviceaufruf.

**Angeben der Origin für den gesamten Prozess**

Tragen Sie die BPMN-Elemente **extensionElements** und **camunda:properties** in die Prozessdefinition ein.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
          ...
      <camunda:property name="service:/process/services/helloworld/
async:callbackBase" value="/" />
      <!-- alternative <camunda:property name="service:*:callbackBase"
value="/" /> -->
      ...
    </camunda:properties>
  </extensionElements>
  ...
</process>
```

Erstellen Sie ein Element **camunda:property** mit dem Namen **service:<service>:callbackBase**. Der Platzhalter **<service>** entspricht der URI des Services, oder * für alle Services.

Die angegebene Origin muss in beiden Fällen dem Format **https://domain[:port]** entsprechen. Alternativ entspricht die Angabe von / einer relativen URI.

**Hinzufügen einer Empfangenaktivität**

Fügen Sie der Prozessdefinition nun die Empfangenaktivität hinzu. Mithilfe der Empfangenaktivität können die verarbeiteten Daten vom Service an den Prozess übermittelt werden.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  ...
  </sendTask>
  <receiveTask id="receive_service_response"
        name="Wait for Service 'Hello World'"
        camunda:asyncAfter="true"
        camunda:exclusive="true">
  </receiveTask>
  ...
</process>
```

**Erläuterung der Eigenschaften:**

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **camunda:***: Diese Eigenschaften beinhalten technische Informationen, die zur Ausführung benötigt werden. Bei asynchronen Services müssen Sie für die Eigenschaft **camunda:asyncAfter** immer den Wert **true** eintragen.

Fügen Sie der Empfangenaktivität die BPMN-Elemente **extensionElements** und **camunda:inputOutput** hinzu. Sie müssen alle Daten definieren, die vom HTTP-Endpunkt des Services als Antwort erwartet werden. Zur Definition der Daten fügen Sie für jeden Wert das BPMN-Element **camunda:outputParameter** ein.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  ...
  </sendTask>
  <receiveTask id="receive_service_response" ...>
    <extensionElements>
      <camunda:inputOutput>
        <camunda:outputParameter name="greeting">$
{variables.get("greeting")}</camunda:outputParameter>
      </camunda:inputOutput>
    </extensionElements>
  </receiveTask>
  ...
</process>
```

**Erläuterung der Parameter:**

- **greeting**: Der Service hat als einzigen Ausgabewert den Parameter **greeting** definiert. Dieser Parameter wird bei der Antwort des Services automatisch in eine gleichnamige Variable im Geltungsbereich der Empfangenaktivität geschrieben. Der Parameter definiert, dass der Inhalt dieses Ergebniswertes bei Fortführung des Prozesses in die Variable **greeting** im Geltungsbereich des gesamten Prozesses geschrieben wird.

Um die Kommunikation zwischen der Anwendung und dem Service zu ermöglichen, müssen Sie die Schnittstelle des Services zum Prozess hinzufügen. Zum Hinzufügen der Schnittstelle tragen Sie, sofern nicht bereits in vorherigen Schritten geschehen, die BPMN-Elemente **extensionElements** und **camunda:properties** in die Prozessdefinition ein. Anschließend fügen Sie je ein BPMN-Element **camunda:property** für den Service-Eingabewert **yourName**, den Service-Ausgabewert **greeting**, sowie die resultierende Prozessvariable **greeting** ein. Alle Werte sind vom Datentyp **String**.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="service:/process/services/helloworld/
async:in:yourName" value="String" />
      <camunda:property name="service:/process/services/helloworld/
async:out:greeting" value="String" />
      <camunda:property name="service:/process/services/helloworld/
async:name" value="Hello World" />
      <camunda:property name="variable:greeting" value="String" />
    </camunda:properties>
  </extensionElements>
```

```
  ...
</process>
```

Die Variable **greeting** enthält nun den Text "Hello <display name of process start user>". Verwenden Sie diese Variable nun als Namen für die Benutzeraktivität, damit der Anwender den freundlichen Gruß als Betreff der zu erledigenden Aufgabe erhält.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  ...
  <userTask id="task_hello_world" name="${variables.get('greeting')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
  ...
</process>
```

Damit die Prozessaktivitäten in der richtigen Reihenfolge ausgeführt werden, müssen Sie die Sequenzflusselemente noch an die neuen Aktivitäten anpassen.

```
...
<process id="async_service_process" name="Async Service Process"
isExecutable="true">
  ...
  <endEvent id="end" />
  <sequenceFlow id="s1" sourceRef="start" targetRef="call_service" />
  <sequenceFlow id="s2" sourceRef="call_service"
targetRef="receive_service_response" />
  <sequenceFlow id="s3" sourceRef="receive_service_response"
targetRef="task_hello_world" />
  <sequenceFlow id="s4" sourceRef="task_hello_world" targetRef="end" />
</process>
```

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI"  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
  <process id="async_service_process" name="Async Service Process"
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="service:/process/services/helloworld/
async:in:yourName" value="String" />
        <camunda:property name="service:/process/services/helloworld/
async:out:greeting" value="String" />
        <camunda:property name="service:/process/services/helloworld/
async:name" value="Hello World" />
        <camunda:property name="variable:greeting" value="String" />
      </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <sendTask id="call_service"
```

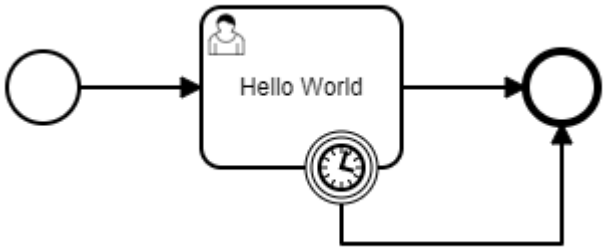
```
          name="Call Service 'Hello World'"
          camunda:asyncBefore="true"
          camunda:delegateExpression="${asyncService}"
          camunda:exclusive="true">
      <extensionElements>
        <camunda:inputOutput>
          <camunda:inputParameter name="service.uri">/process/services/
helloworld/async</camunda:inputParameter>
          <camunda:inputParameter name="yourName">$
{variables.getDisplayValue("dv_initiator")}</camunda:inputParameter>
        </camunda:inputOutput>
      </extensionElements>
    </sendTask>
    <receiveTask id="receive_service_response"
          name="Wait for Service 'Hello World'"
          camunda:asyncAfter="true"
          camunda:exclusive="true">
      <extensionElements>
        <camunda:inputOutput>
          <camunda:outputParameter name="greeting">$
{variables.get("greeting")}</camunda:outputParameter>
        </camunda:inputOutput>
      </extensionElements>
    </receiveTask>
    <userTask id="task_hello_world" name="${variables.get('greeting')}"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="call_service" />
    <sequenceFlow id="s2" sourceRef="call_service"
targetRef="receive_service_response" />
    <sequenceFlow id="s3" sourceRef="receive_service_response"
targetRef="task_hello_world" />
    <sequenceFlow id="s4" sourceRef="task_hello_world" targetRef="end" />
  </process>

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="async_service_process">
      <bpmndi:BPMNEdge id="s4_di" bpmnElement="s4">
        <di:waypoint x="630" y="117" />
        <di:waypoint x="662" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="490" y="117" />
        <di:waypoint x="530" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="350" y="117" />
        <di:waypoint x="390" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="117" />
        <di:waypoint x="250" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
```

```
        <dc:Bounds x="179" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="call_service_di" bpmnElement="call_service">
        <dc:Bounds x="250" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="receive_service_response_di"
bpmnElement="receive_service_response">
        <dc:Bounds x="390" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="530" y="77" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="662" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

### 1.4.12. Verwenden von Eskalationen

Beim Modellieren eigener Prozesse können Sie Eskalationen verwenden, um den Prozessablauf zu beeinflussen. Zum Einstieg in das Arbeiten mit Eskalationen ist es hilfreich, wenn Sie zunächst einen einfachen Prozess mit einer Benutzeraktivität und einem Timer-Ereignis erstellen. Dieser Prozess sieht beispielsweise so aus:




Das BPMN-Modell dieses beispielhaften Prozesses ist folgendermaßen aufgebaut:

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="escalation_process" name="Escalation Process"
isExecutable="true">
    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <boundaryEvent id="timer" attachedToRef="task_hello_world">
      <timerEventDefinition>
        <timeDuration>PT1M</timeDuration>
      </timerEventDefinition>
    </boundaryEvent>
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <sequenceFlow id="s3" sourceRef="timer" targetRef="end" />
```

```
  </process>
</definitions>
```

Der Beispielprozess soll nun so angepasst werden, dass beim Eintritt des Timer-Ereignisses eine Eskalation ausgelöst wird. Die Eskalation startet dann einen Subprozess, der den Eskalationspfad beschreibt. Für diese Anpassung erzeugen Sie zunächst ein Eskalationsobjekt.

**Hinzufügen eines Eskalationsobjektes**

Fügen Sie dem Prozess zunächst das BPMN-Element **escalation** hinzu.

```
...
<definitions ...>
  <process id="escalation_process" name="Escalation Process"
isExecutable="true">
    ...
  </process>
  <escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />
</definitions>
```

**Erläuterung der Eigenschaften:**

- **id:**Die Eigenschaft wird als eindeutiger Bezeichner für die Eskalation verwendet.
- **name:** Der Anzeigename der Eskalation.
- **escalationCode:** Die Eigenschaft definiert einen technischen Code, der von den auf Eskalationen wartende Elementen verwendet werden kann, um die auslösende Eskalation zu identifizieren.

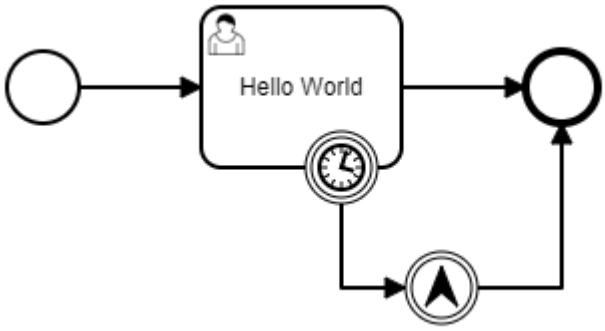
Nun kann die Eskalation im BPMN-Element **IntermediateThrowEvent** oder einem **EndEvent** verwendet werden, um ein entsprechendes Eskalationsereignis auszulösen.

**Hinzufügen eines Eskalationsereignisses**

Fügen Sie dem Prozess zunächst das BPMN-Element **IntermediateThrowEvent** mit einem **EscalationEventDefinition**-Element hinzu. Anschließend ändern Sie das Ziel des Sequenzflusses **s3** von **end** nach **throw_time_escalation** und fügen einen neuen Sequenzfluss **s4** von **throw_time_escalation** nach **end** hinzu.

```
...
<process id="escalation_process" name="Escalation Process"
isExecutable="true">
  ...
  <intermediateThrowEvent id="throw_time_escalation">
    <escalationEventDefinition escalationRef="time_escalation" />
  </intermediateThrowEvent>
  <endEvent id="end" />
  ...
  <sequenceFlow id="s3" sourceRef="timer" targetRef="throw_time_escalation"
/>
  <sequenceFlow id="s4" sourceRef="throw_time_escalation" targetRef="end" />
</process>
<escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />
...
```

Der Prozess sieht nun wie folgt aus:




**Hinzufügen eines Eskalationspfades**

Fügen Sie nun einen Subprozess hinzu, dessen BPMN-Element **StartEvent** das Element **EscalationEvent-Definition** beinhaltet.

```
...
<process id="escalation_process" name="Escalation Process"
isExecutable="true">
  <extensionElements>
    <camunda:properties>
      <camunda:property name="variable:escalationCode" value="String" />
    </camunda:properties>
  </extensionElements>
  <startEvent id="start" />
  ...
  <sequenceFlow id="s4" sourceRef="throw_time_escalation" targetRef="end" />
  <subProcess id="escalation_subprocess" name="Escalation Process"
triggeredByEvent="true">
    <startEvent id="sub_start" isInterrupting="false">
      <escalationEventDefinition escalationRef="time_escalation"
camunda:escalationCodeVariable="escalationCode" />
    </startEvent>
    <endEvent id="sub_end" />
    <sequenceFlow id="sub_s1" sourceRef="sub_start" targetRef="sub_end" />
  </subProcess>
</process>
<escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />
...
```

**Erläuterung der Eigenschaften:**

- **subProcess**
  - **triggeredByEvent**: muss den Wert **true** enthalten. Die Eigenschaft gibt an, dass das Start-Ereignis dieses Subprozesses durch ein Ereignis ausgelöst wird (in diesem Fall ein Eskalationsereignis).
  - **isInterrupting**: wenn **true**, dann wird der Prozesszweig, der die Eskalation ausgelöst hat, abgebrochen. Wenn **false**, dann läuft dieser Prozesszweig parallel weiter.
- **escalationEventDefinition**
  - **escalationRef:** Enthält die ID der Eskalation, anhand derer dieses Ereignis ausgelöst werden soll.
  - **camunda:escalationCodeVariable:** (Optional) Name der Variablen, in die der Wert von **escalationCode** der aufgetretenen Eskalation gespeichert wird (Variable muss definiert sein).

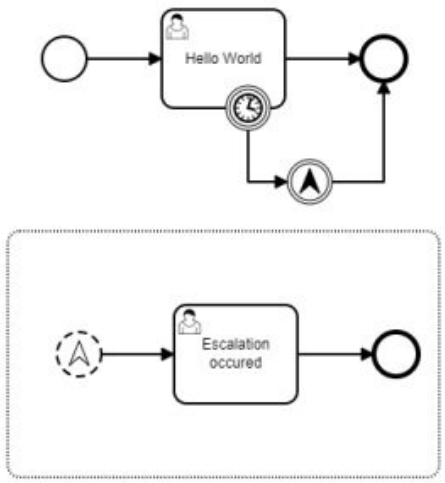
Bitte beachten Sie, dass Sie bei Verwendung der Eigenschaft **camunda:escalationCodeVariable** auch eine Definition der entsprechenden Prozessvariablen im Hauptprozess hinzufügen müssen.

Fügen Sie dem Eskalationspfad nun noch eine Benutzeraktivität hinzu, die im Falle einer Eskalation durchlaufen werden soll.

```
...
<process id="escalation_process" name="Escalation Process"
isExecutable="true">
  ...
  <subProcess id="escalation_subprocess" name="Escalation Process"
triggeredByEvent="true">
    <startEvent id="sub_start" isInterrupting="false">
      <escalationEventDefinition escalationRef="time_escalation"
camunda:escalationCodeVariable="escalationCode" />
    </startEvent>
        <userTask id="sub_task_escalation" name="Escalation occurred"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <endEvent id="sub_end" />
    <sequenceFlow id="sub_s1" sourceRef="sub_start"
targetRef="sub_task_escalation" />
    <sequenceFlow id="sub_s2" sourceRef="sub_task_escalation"
targetRef="sub_end" />
  </subProcess>
</process>
<escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />
...
```

Zur Vereinfachung wird in diesem Beispiel als Empfänger der Benutzeraktivität des Eskalationspfades der Benutzer eingetragen, der den Prozess gestartet hat.

Der finale Prozess sieht wie folgt aus:




Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus.

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI"  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
  <process id="escalation_process" name="Escalation Process"
```

```
isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:escalationCode" value="String" />
      </camunda:properties>
    </extensionElements>

    <startEvent id="start" />
    <userTask id="task_hello_world" name="Hello World"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
    <boundaryEvent id="timer" attachedToRef="task_hello_world">
      <timerEventDefinition>
        <timeDuration>PT1M</timeDuration>
      </timerEventDefinition>
    </boundaryEvent>
    <intermediateThrowEvent id="throw_time_escalation">
      <escalationEventDefinition escalationRef="time_escalation" />
    </intermediateThrowEvent>
    <endEvent id="end" />

    <sequenceFlow id="s1" sourceRef="start" targetRef="task_hello_world" />
    <sequenceFlow id="s2" sourceRef="task_hello_world" targetRef="end" />
    <sequenceFlow id="s3" sourceRef="timer"
targetRef="throw_time_escalation" />
    <sequenceFlow id="s4" sourceRef="throw_time_escalation" targetRef="end"
/>

    <subProcess id="escalation_subprocess" name="Escalation Process"
triggeredByEvent="true">
      <startEvent id="sub_start" isInterrupting="false">
        <escalationEventDefinition escalationRef="time_escalation"
camunda:escalationCodeVariable="escalationCode" />
      </startEvent>
      <userTask id="sub_task_escalation" name="Escalation occurred"
camunda:candidateUsers="${variables.get('dv_initiator')}" />
      <endEvent id="sub_end" />
      <sequenceFlow id="sub_s1" sourceRef="sub_start"
targetRef="sub_task_escalation" />
      <sequenceFlow id="sub_s2" sourceRef="sub_task_escalation"
targetRef="sub_end" />
    </subProcess>

  </process>
  <escalation id="time_escalation" name="Time Escalation"
escalationCode="123" />

  <!-- Diagram information -->
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="escalation_process">
      <bpmndi:BPMNShape id="start_di" bpmnElement="start">
        <dc:Bounds x="179" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="task_hello_world_di"
bpmnElement="task_hello_world">
        <dc:Bounds x="290" y="77" width="100" height="80" />
```

```
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="timer_di" bpmnElement="timer">
        <dc:Bounds x="372" y="139" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="throw_time_escalation_di"
bpmnElement="throw_time_escalation">
        <dc:Bounds x="432" y="202" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="end_di" bpmnElement="end">
        <dc:Bounds x="492" y="99" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="s1_di" bpmnElement="s1">
        <di:waypoint x="215" y="117" />
        <di:waypoint x="290" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s2_di" bpmnElement="s2">
        <di:waypoint x="390" y="117" />
        <di:waypoint x="492" y="117" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s3_di" bpmnElement="s3">
        <di:waypoint x="390" y="175" />
        <di:waypoint x="390" y="220" />
        <di:waypoint x="432" y="220" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="s4_di" bpmnElement="s4">
        <di:waypoint x="468" y="220" />
        <di:waypoint x="510" y="220" />
        <di:waypoint x="510" y="135" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="escalation_subprocess_di"
bpmnElement="escalation_subprocess" isExpanded="true">
        <dc:Bounds x="165" y="290" width="350" height="200" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="sub_start_di" bpmnElement="sub_start">
        <dc:Bounds x="205" y="372" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="sub_task_escalation_di"
bpmnElement="sub_task_escalation">
        <dc:Bounds x="290" y="350" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="sub_end_di" bpmnElement="sub_end">
        <dc:Bounds x="442" y="372" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="sub_s1_di" bpmnElement="sub_s1">
        <di:waypoint x="241" y="390" />
        <di:waypoint x="290" y="390" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="sub_s2_di" bpmnElement="sub_s2">
        <di:waypoint x="390" y="390" />
        <di:waypoint x="442" y="390" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

### 1.4.13. Verändern von Prozessvariablen einer Multi-Instanz

Innerhalb von Subprozessen, die als Multi-Instanz konfiguriert sind, ist das Verändern von Prozessvariablen zunächst nur innerhalb des Geltungsbereichs der aktuellen Subprozessinstanz möglich.

Das folgende Prozessbeispiel zeigt einen Prozess, in dem einer Gruppe von Empfängern (Variable **allAssignees**) je eine Benutzeraufgabe zugewiesen wird. Hierfür wird eine Multi-Instanz verwendet. Im Rahmen der Benutzeraufgabe soll jeder Benutzer z.B. mithilfe eines Formulars eine Variable **decision** einfügen. Da jede Subprozessinstanz über ihren eigenen Geltungsbereich der Variablen verfügt, ist eine Zusammenfassung der jeweils in **decision** eingetragenen Werte aller Empfänger derzeit nicht möglich.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" targetNamespace=""
exporter="d.velop process modeler">
  <process id="writeMultiInstanceVariables"
name="WriteMultiInstanceVariables" isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:allAssignees*" value="[Identity]!"
/>
        <camunda:property name="variable:assignee" value="Identity" />
        <camunda:property name="variable:allDecisions" value="[String]" />
        <camunda:property name="variable:decision" value="String" />
          </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="subflow" />
    <subProcess id="subflow">
      <multiInstanceLoopCharacteristics camunda:collection="$
{variables.get('assignees')}" camunda:elementVariable="assignee" />
      <startEvent id="subflow_start" />
      <sequenceFlow id="s2-1" sourceRef="subflow_start"
targetRef="subflow_usertask" />
      <endEvent id="subflow_end" />
      <sequenceFlow id="s2-2" sourceRef="subflow_usertask"
targetRef="subflow_end" />
      <userTask id="subflow_usertask" name="Decision"
camunda:candidateUsers="${variables.get('assignee')}" />
    </subProcess>
    <sequenceFlow id="s3" sourceRef="subflow" targetRef="end" />
  </process>
</definitions>
```

Um Prozessvariablen aus dem Geltungsbereich der Multi-Instanz heraus in den Geltungsbereich des gesamten Prozesses zu schreiben, müssen Sie dem Subprozess einen Schritt vom Typ **bpmn:serviceTask** hinzufügen.

```
...
 <subProcess id="subflow">
   ...
   <sequenceFlow id="s2-3" sourceRef="writeDecision"
targetRef="subflow_end" />
   <serviceTask id="writeDecision" name="Write decision to
global scope" camunda:asyncBefore="true" camunda:asyncAfter="true"
```

```
camunda:delegateExpression="${writeMultiInstanceVariables}">
     <extensionElements>
       ...
       <camunda:inputOutput>
         <camunda:inputParameter name="allDecisions">$
{variables.get('decision')}</camunda:inputParameter>
       </camunda:inputOutput>
     </extensionElements>
   </serviceTask>
</subProcess>
...
```

**Erläuterung der Eigenschaften**

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für die Sendenaktivität verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **camunda:***: Diese Eigenschaften beinhalten technische Informationen, die zur Ausführung benötigt werden. Bei Schritten zum Schreiben von Variablen aus einer Multi-Instanz in den Prozess müssen Sie für diese Eigenschaften immer die Werte des Beispiels eintragen.

Im Bereich **camunda:inputOutput** müssen Sie nun für jede Variable, für die Sie im Geltungsbereich des Prozesses einen Wert einfügen wollen, ein Element vom Typ **camunda:inputParameter** definieren. Die Eigenschaft **name** muss in diesem Fall dem Namen der Prozessvariablen entsprechen, in die geschrieben werden soll. Der Inhalt dieses Elements definiert den Wert, der geschrieben wird. Hier können Sie wie gewohnt auch auf Variablen (des Geltungsbereichs der Multi-Instanz oder auch des Gesamtprozesses) zugreifen.

Prozessvariablen, die auf diese Weise geschrieben werden, müssen immer vom Typ **Mehrfachwert** sein. Der Index, an den eine Instanz des Multi-Instanz-Subprozesses schreibt, wird durch den automatisch zugewiesenen Wert der Variablen **loopCounter** bestimmt.

> **Anmerkung**
> **allAssignees** hat die Werte *['user1', 'user2', 'user3']*. Dadurch entstehen insgesamt drei Instanzen des Multi-Instanz-Subprozesses mit den folgenden (hier relevanten) Variablen:
>
> Instanz 1:
>
> assignee: *'user1'*
>
> loopCounter: *0*
>
> Instanz 2:
>
> assignee: *'user2'*
>
> loopCounter: *1*
>
> Instanz 3:
>
> assignee: *'user3'*
>
> loopCounter: *2*
>
> Würde nun *user2* in Instanz 2 die Benutzeraufgabe mit dem Wert *'myDecision'* für **decision** abschließen und im folgenden Schritt die Variable **allDecisions** schreiben, hätte diese im Anschluss folgende Werte:
>
> allDecisions: *[null, 'myDecision', null]*

Damit das BPMN-Diagramm im Modellierungswerkzeug sowie der Benutzeroberfläche korrekt dargestellt werden kann, wurde es noch um Diagramminformationen erweitert. Die finale BPMN-Definition sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/
20100524/DI"  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
targetNamespace="" exporter="d.velop process modeler">
 <process id="writeMultiInstanceVariables"
name="WriteMultiInstanceVariables" isExecutable="true">
    <extensionElements>
      <camunda:properties>
        <camunda:property name="variable:allAssignees*" value="[Identity]!"
/>
        <camunda:property name="variable:assignee" value="Identity" />
        <camunda:property name="variable:decision" value="String" />
        <camunda:property name="variable:allDecisions" value="[String]" />
      </camunda:properties>
    </extensionElements>
    <startEvent id="start" />
    <endEvent id="end" />
    <sequenceFlow id="s1" sourceRef="start" targetRef="subflow" />
    <subProcess id="subflow">
      <multiInstanceLoopCharacteristics camunda:collection="$
{variables.get('allAssignees')}" camunda:elementVariable="assignee" />
      <startEvent id="subflow_start" />
      <sequenceFlow id="s2-1" sourceRef="subflow_start"
targetRef="subflow_usertask" />
      <endEvent id="subflow_end" />
      <sequenceFlow id="s2-2" sourceRef="subflow_usertask"
targetRef="writeDecision" />
      <userTask id="subflow_usertask" name="Decision"
camunda:candidateUsers="${variables.get('assignee')}" />
      <sequenceFlow id="s2-3" sourceRef="writeDecision"
targetRef="subflow_end" />
      <serviceTask id="writeDecision" name="Write decision to
global scope" camunda:asyncBefore="true" camunda:asyncAfter="true"
camunda:delegateExpression="${writeMultiInstanceVariables}">
        <extensionElements>
          <camunda:failedJobRetryTimeCycle>R3/PT5M</
camunda:failedJobRetryTimeCycle>
          <camunda:inputOutput>
            <camunda:inputParameter name="allDecisions">$
{variables.get('decision')}</camunda:inputParameter>
          </camunda:inputOutput>
        </extensionElements>
      </serviceTask>
    </subProcess>
    <sequenceFlow id="s3" sourceRef="subflow" targetRef="end" />
  </process>

  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1"
bpmnElement="writeMultiInstanceVariables">
```

```
      <bpmndi:BPMNShape id="_BPMNShape_StartEvent_2" bpmnElement="start">
        <dc:Bounds x="179" y="159" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Event_0ssel47_di" bpmnElement="end">
        <dc:Bounds x="912" y="159" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_0li7yi3_di" bpmnElement="subflow"
isExpanded="true">
        <dc:Bounds x="300" y="77" width="490" height="200" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Event_0ybst54_di" bpmnElement="subflow_start">
        <dc:Bounds x="340" y="159" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Event_0f9450b_di" bpmnElement="subflow_end">
        <dc:Bounds x="712" y="159" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_1qx66xt_di"
bpmnElement="subflow_usertask">
        <dc:Bounds x="430" y="137" width="100" height="80" />
        <bpmndi:BPMNLabel />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_13t76uu_di"
bpmnElement="writeDecision">
        <dc:Bounds x="570" y="137" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="Flow_146y71c_di" bpmnElement="s2-1">
        <di:waypoint x="376" y="177" />
        <di:waypoint x="430" y="177" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1yq9qcz_di" bpmnElement="s2-2">
        <di:waypoint x="530" y="177" />
        <di:waypoint x="570" y="177" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1huut97_di" bpmnElement="s2-3">
        <di:waypoint x="670" y="177" />
        <di:waypoint x="712" y="177" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1o99dar_di" bpmnElement="s1">
        <di:waypoint x="215" y="177" />
        <di:waypoint x="300" y="177" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_13w0opv_di" bpmnElement="s3">
        <di:waypoint x="790" y="177" />
        <di:waypoint x="912" y="177" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

### 1.4.14. Verändern von Prozessvariablen

Um Prozessvariablen zu schreiben, müssen Sie dem Subprozess einen Schritt vom Typ **bpmn:serviceTask** hinzufügen.

```
...
 <process id="my-process">
```

```
   ...
   <serviceTask id="writeProcessVariables" name="Write process variables"
camunda:delegateExpression="${writeGlobalVariables}">
     <extensionElements>
       ...
       <camunda:inputOutput>
         <camunda:inputParameter name="textVariable">someValue</
camunda:inputParameter>
         <camunda:inputParameter name="numberVariable">${123}</
camunda:inputParameter>
           </camunda:inputOutput>
     </extensionElements>
   </serviceTask>
</process>
...
```

**Erläuterung der Eigenschaften**

- **id**: Die Eigenschaft wird als eindeutiger Bezeichner für die Serviceaktivität verwendet.
- **name**: Die Eigenschaft wird als Anzeigename in den Benutzeroberflächen verwendet.
- **camunda:delegateExpression**: **${writeLocalVariables}**, wenn die Variablen in den lokalen Geltungsbereich, z.B. eines Sub-Prozesses, geschrieben werden sollen. **${writeGlobalVariables}**, wenn die Variablen in den Geltungsbereich des Prozesses geschrieben werden sollen.

Im Bereich **camunda:inputOutput** müssen Sie nun für jede Variable, für die Sie einen Wert einfügen wollen, ein Element vom Typ **camunda:inputParameter** definieren. Die Eigenschaft **name** muss in diesem Fall dem Namen der Prozessvariablen entsprechen, in die geschrieben werden soll. Der Inhalt dieses Elements definiert den Wert, der geschrieben wird. Unabhängig davon, ob Sie **${writeLocalVariables}** oder **${writeGlobalVariables}** verwenden, muss die Variable direkt im Prozess definiert worden sein.

## 1.5. Häufig gestellte Fragen

Sie finden in diesem Thema Antworten zu häufig gestellten Fragen.

### 1.5.1. Warum tritt beim Einfügen einer Variablen ein unbekannter Serverfehler auf?

Wenn Sie eine Prozessvariable einfügen wollen, die bereits in einer anderen Groß-/Kleinschreibung existiert, kann es zu einem Fehler kommen. Der Fehler tritt auf, wenn die Sortierung der Datenbank die Groß-/Kleinschreibung nicht beachtet und somit nicht zwischen den beiden Variablen unterscheiden kann.

Zur Vermeidung des Fehlers verwenden Sie beim Einfügen einer Variablen die bereits verwendete Groß-/Kleinschreibung.

### 1.5.2. Wo finde ich eine Übersicht über alle Aktionen?

Wenn Sie im Feature **Prozessadministration** in die Perspektive **Aktionsübersicht** wechseln, erhalten Sie einen Überblick über alle laufenden, gestarteten, abgeschlossenen und fehlgeschlagenen Aktionen. Mit dem Papierkorb-Symbol können Sie abgeschlossene und fehlerhafte Aktionen manuell aus dem Protokoll entfernen.

## 1.6. Weitere Informationsquellen und Impressum

Wenn Sie Ihre Kenntnisse rund um die d.velop-Software vertiefen möchten, besuchen Sie die digitale Lernplattform der d.velop academy unter https://dvelopacademy.keelearning.de/.

Mithilfe der E-Learning-Module können Sie sich in Ihrem eigenen Tempo weiterführende Kenntnisse und Fachkompetenz aneignen. Zahlreiche E-Learning-Module stehen Ihnen ohne vorherige Anmeldung frei zugänglich zur Verfügung.

Besuchen Sie unsere Knowledge Base im d.velop service portal. In der Knowledge Base finden Sie die neusten Lösungen, Antworten auf häufig gestellte Fragen und How To-Themen für spezielle Aufgaben. Sie finden die Knowledge Base unter folgender Adresse: https://kb.d-velop.de/

Das zentrale Impressum finden Sie unter https://www.d-velop.de/impressum.