Action

Plugins haben die Möglichkeit, Actionbausteine bereitzustellen, die ganz normal in Menüpunkt-Aktionen im Framework verwendet werden können. Plugin-Actionbausteine stehen dann dort (gruppiert nach Plugin) genau wie die Standard-Bausteine zur Auswahl.

```
fun registerAction(actionId: String, actionInfo: ActionInfo)
```

`actionId` ist eine frei wählbare ID, die über alle Actionbausteine eines Plugins hinweg eindeutig sein muss. Es ist kein Problem, wenn mehrere Plugins einen Baustein mit derselben ID bereitstellen, da zum Ansprechen eines Actionbausteins auch die `id` des Plugins herangezogen wird.

`actionInfo` enthält die Metadaten des Actionbausteins, wie Titel und Beschreibung. Es kann auch ein Link zu einer externen Hilfeseite definiert werden. Außerdem wird die Klasse referenziert, welche den Baustein implementiert.

Diese Klasse muss von `com.batix.plugins.PluginAction` abgeleitet sein und die `doAction()` Methode implementieren. Jeder Aufruf einer Aktion erzeugt für jeden Actionbaustein eine eigene Instanz der entsprechenden Klasse. Innerhalb von `doAction()` stehen dann z. B. `request` und `response` zur Verfügung.

Parameter / Properties

Damit der Baustein in der Verwaltung konfiguriert werden kann, müssen dessen Parameter definiert werden. Diese werden in `actionInfo` hinterlegt und können dann in `doAction()` ausgelesen werden.

Es können verschiedenartige Parameter hinterlegt werden, so gibt es z. B. die Methoden `addTextParameter` für Texteingaben oder `addBooleanParameter` für eine Ja/Nein-Auswahl.

```
fun addTextParameter(
  name: String,
  title: String,
  description: String,
  descriptionIsHtml: Boolean,
  required: Boolean
)
```

`name` ist ein interner Bezeichner, der dann auch wieder beim Auslesen in `doAction` angegeben werden muss.

> **Tip**
> Es empfiehlt sich, diesen internen Wert als Konstante im Code zu definieren, da der Wert an mehreren Stellen im Code (Parameterdefinition und Auslesen) benötigt wird.

Mit `title` kann für die Anzeige im Backend ein freundlicher Name gesetzt werden. `description` ist ein Erklärungstext, der unter dem Titel angezeigt wird – `descriptionIsHtml` legt fest, ob dieser bereits als HTML formatiert ist oder nicht. Falls `required` gesetzt ist, wird dieser Parameter in der Verwaltung als Pflichteingabe markiert.

In der Action kann dann mit `getProperty`, `getPropertyReplaced` oder `getBooleanProperty` der vom Benutzer eingestellte Wert abgefragt werden.

```
fun getProperty(key: String): String?
fun getProperty(key: String, defaultValue: String?): String?

fun getPropertyReplaced(key: String): String?
fun getPropertyReplaced(key: String, defaultValue: String?): String?

fun getBooleanProperty(key: String): Boolean
```

`getPropertyReplaced` ersetzt Platzhalter in der Form [[paramname]] durch den Wert des Request-Parameters oder Actionscript-Attributs `paramname`.

Mittels `addCustomParameter` kann sogar komplett eigener HTML-Code zur Benutzereingabe geliefert werden. Dazu leitet man am besten eine Klasse von `com.batix.action.ExtendedActionParameter` ab und implementiert im Minimum die folgenden Methoden:

- `getName()` – liefert den internen Bezeichner zurück
- `getTitle()` – gibt den Titel zurück
- `writeAdminView(StringBuffer, String, Connection)`
  - in den `StringBuffer` schreibt man den HTML-Quelltext, der in der Verwaltung angezeigt werden soll
  - der `String` ist der aktuell gespeicherte Wert des Parameters
  - `Connection` ist eine Datenbankverbindung

Wichtig hierbei ist, dass `writeAdminView` ein HTML-Formular-Element mit passendem `name` erzeugt (Prefix `param:` und dann der interne Bezeichner), damit die Eingabe auch gespeichert wird.

Beispiel

Das Beispiel-Action definiert 3 Parameter: einen vom Typ Text, eine Ja/Nein-Auswahl und einen speziellen Parameter. Die Action-Klasse definiert die internen Bezeichner der Parameter als Konstanten. In der `doAction`-Methode werden die Parameter ausgelesen.

```
import com.batix.plugins.PluginAction

class ImportMitarbeiterAction : PluginAction() {
  companion object {
    const val PROP_API_TOKEN = "api-token"
    const val PROP_SEND_NOTIFICATION = "send-notification"
    const val PROP_IMPORTANT_THING = "important-thing"
  }

  override fun doAction() {
    val apiToken = getPropertyReplaced(PROP_API_TOKEN, "")
    require(!apiToken.isNullOrEmpty()) { "API-Token darf nicht leer sein" }

    val doSendNotification = getBooleanProperty(PROP_SEND_NOTIFICATION)

    val importantThing = getProperty(PROP_IMPORTANT_THING)

    // ...
  }
}
```

Damit der Baustein in der Verwaltung auftaucht, muss dieser dem Application Framework bekannt gegeben werden. Das erfolgt in der `load`-Methode der Plugin-Hauptklasse. Hier werden nebst den Metadaten des Bausteins auch dessen Parameter definiert.

```
override fun load() {
  registerAction(
    "import-mitarbeiter", // actionId
    ActionInfo(
      "Mitarbeiter importieren", // title
      "Importiert Mitarbeiter aus der externen Datenquelle.", // description
      false, // deprecated
      true, // beta
      "https://www.batix.de/unternehmen/unternehmenskultur/team/", // helpLink
      ImportMitarbeiterAction::class.java // actionClass
    ).apply {
      addTextParameter(
```

```
        ImportMitarbeiterAction.PROP_API_TOKEN, // name
        "API Token", // title
        "Token zur Authentifizierung an der abzufragenden API", // description
        false, // descriptionIsHtml
        true // required
      )

      addBooleanParameter(
        ImportMitarbeiterAction.PROP_SEND_NOTIFICATION, // name
        "Benachrichtigung versenden", // title
        "(Standard: nein)", // description
        false, // descriptionIsHtml
        false // required
      )

      addCustomParameter(object : ExtendedActionParameter() {
        override fun getName(): String {
          return ImportMitarbeiterAction.PROP_IMPORTANT_THING
        }

        override fun getTitle(): String {
          return "Wichtige Einstellung"
        }

        override fun writeAdminView(sb: StringBuffer, value: String?, conn: Connection) {
          sb.append("""<input type="text" """)
          sb.append("""name="param:${ImportMitarbeiterAction.PROP_IMPORTANT_THING}" """)
          sb.append("""value="${Tools.htmlEncode(value ?: "")}" """)
          sb.append("""style="border: 1px solid red;">""")
        }
      })
    }
  )
}
```

Der Plugin-Actionbaustein taucht somit in der Auswahl der Actionbausteine auf.

API Token: (api-token)

Token zur Authentifizierung an der abzufragenden API

Benachrichtigung versenden: (send-notification)

(Standard: nein)

ja nein

Wichtige Einstellung:

Freieingabe weiterer Eigenschaften