CMS-spezifische Helfer

Bestimmte CMS-Klassen wurden um Utilities zur einfacheren Verwendung erweitert.

Importe

Einige häufig genutzte Klassen werden automatisch importiert, z.B. `java.sql.Connection`, `TableMetadata` und `BatixRecord`.

Objekte

Dem Groovy-Script stehen automatisch folgende Objekte zur Verfügung:

- `includeTag` - Referenz zum ausführenden `bx:groovy` Tag, ansonsten `null`
  - es können hier z.B. Tag-Parameter gelesen werden:
    Template

    ```
    <bx:groovy includes="MyReport" format="html" />
    ```

    Groovy-Baustein

    ```
    String format = includeTag.getStringParameter("format")
    println("gewähltes Format: $format")
    ```

- `action` - Referenz zum ausführenden Groovy-Action, ansonsten `null`
  - kann z.B. benutzt werden, um Script-Attribute zu setzen oder das Action abzubrechen

    ```
    action.setScriptAttribute("thing", "value")
    action.cancel = true
    ```

- `application` - der `ServletContext`
- `request` - der aktuelle `HttpServletRequest`
  - z.B. um Request-Parameter auszulesen, die der Seite oder dem Action übergeben wurden

    ```
    String kid = request.getParameter("kid")
    ```

- `response` - die aktuelle `HttpServletResponse`

    - falls ein Action verwendet wird: `action.originalResponse` benutzen, um z.B. direkt Bytes zu schreiben
- `session` - die aktuelle Session, falls vorhanden
- `pageData` - enthält Infos zum Aufruf

```
String file = pageData.filename // z.B. "detail.htm"
```

- `navElement` - der aufgerufene Menüpunkt (`NavigationElement`)
- `web` - das Projekt (`Customer`)
- `variables` - die Systemeinstellungen (`SystemVariables`)
    - Das sind die pro Projekt überschreibbaren Variablen

```
String portalId = variables.getVariable("PortalID")
```

Connection

Um einfach an eine Datenbank-Connection zu kommen, egal ob der Groovy-Baustein in einem Action oder Tag (oder sogar den Entwickler-Tools) verwendet wird, kann folgender Block verwendet werden:

```
Connection.with { conn ->
  // hier Code, der conn benutzt
}
```

Die Connection wird nach dem Block automatisch wieder geschlossen bzw. zurückgegeben.

TableMetadata holen

Eine Instanz von `TableMetadata` lässt sich einfach zu einem Container erzeugen, in dem Entweder der Titel, der Tabellenname oder die ID des Containers benutzt wird (es wird eine `TableMetadata`-Instanz zurückgegeben):

```
// funktioniert alles gleich
def tmdKunden = TableMetadata["Kunden"]
def tmdKunden = TableMetadata["bxc_kunden"]
def tmdKunden = TableMetadata["15525037CCA"]

// auch mit Punkt-Schreibweise verfügbar
def tmdKunden = TableMetadata.Kunden
def tmdKunden = TableMetadata.bxc_kunden
```

```
def tmdKunden = TableMetadata."15525037CCA" // als String, da sonst Syntax Error wegen Ziffer am Anfang
```

BatixRecord holen

Ein bestimmter Datensatz aus einem Container, kann durch seine ID einfach geholt werden (es wird eine `BatixRecord`-Instanz zurückgegeben):

```
def recKunde = tmdKunden["15525037CCB"]
```

Einen neuen Datensatz erzeugt man mit den üblichen Methoden:

```
def recKunde = tmdKunden.createNewRecord(conn, com.batix.util.UniqueID.createHexId(), true)
```

Datensatz-Felder lesen / schreiben

Hat man einen `BatixRecord` kann man auf einfache Art und Weise die Datensatz-Felder lesen und ändern:

```
// Text
String name = recKunde.Name.string
recKunde.Titel.string = "Dr."

// Zahl
int alter = recKunde.Alter.number // Ganzzahl-Feld
recKunde.Alter.number = 18
double rabatt = recKunde.Rabatt.number // Dezimalzahl-Feld
recKunde.Rabatt.number = 5.83

// Datum, Datum + Uhrzeit, Uhrzeit oder Zeitraum
Date bday = recKunde.Geburtstag.date
recKunde.lastUpdate.date = new Date()

// Häkchen
boolean archived = recKunde.archiviert.state
recKunde.Newsletter.state = false

// Einzelverknüpfung (Bild, Dokument, Datensatz)
```

```
String picId = recKunde.Foto.string
String statusId = recKunde.Status.string
recKunde.Rolle.string = "15525037CCC"

// Mehrfachverknüpfung
MultipleLinkField leistungen = recKunde.Leistungen

// Untercontainer
SubListField vertraege = recKunde.Vertraege

// zum Schluss Update nicht vergessen
recKunde.updateRecordInDatabase(conn)
// oder Anlegen, falls es ein neuer Datensatz war
recKunde.createRecordInDatabase(conn, false) // false = Default-Werte der Container-Felder
übernehmen
```

> Durch die angesprochenen Probleme mit Typ-Infos, kann es passieren, dass bei Datum/Zeit-Feldern der Setter uneindeutig ist, falls `null` übergeben wird (da es dort mehrere `setDate` Methoden gibt).
>
> Falls eine Variable übergeben wird, die auch `null` sein kann, wird daher beim Setzen von Datum/Zeit Feldern empfohlen, die `setDate` Methode aufzurufen und einen expliziten Cast hinzuzufügen:

```
Date birthday = ...
recKunde.Geburtstag.setDate(birthday as Date)
```

Mehrere Datensätze filtern

Ähnlich dem XML-Filter mit bx:containerfilter können in Groovy Datensätze gefiltert werden. Der Methode werden als erster Parameter eine Liste an Kriterien (bestehend aus einer drei-Elementigen Liste pro Kriterium mit: Feld(ern), Vergleichstyp und Vergleichswert) und als zweiter Parameter eine Map von Optionen übergeben. Das Ergebnis is ein (evtl. leeres) Array aus `BatixRecord`s.

Ab Version 2.6.8 kann bei der Kriterien-Liste jeweils ein vierter Parameter angegeben werden (`true` oder `false`), welcher dem `required`-Attribut in der XML entspricht. Das ist nützlich, falls man direkt Sachen aus dem Request o.ä. als Vergleichswert angibt: `["Name", 1, request.getParameter("name"), true]` (in späteren Versionen führt ein leerer Vergleichswert ohne explizite required-Angabe dann zum Abbruch)

```
BatixRecord[] recs = tmdKunden.findRecords([
  ["Leistungen", 5, 0], // Feld darf nicht leer sein
  ["Newsletter", 4, 1], // Feld muss angehakt sein
  [["Name", "Vorname"], 1, "muster"] // mindestens eins der beiden Felder muss "muster"
enthalten
], [
  orderby: "Kundennummer", // optional, Sortierungsfeld (Standard ID)
  desc: true, // optional, Sortierrichtung (Standard aufsteigend)
  index: 5, // optional, Startindex (Null-basiert)
  limit: 20, // optional, maximale Anzahl Ergebnisse
  active: true, // optional, aktive Datensätze suchen? (Standard ja)
  inactive: true // optional, inaktive Datensätze suchen? (Standard nein)
])
```

Die Zahlen entsprechen den Typen des normalen Filters. Es müssen nicht alle Parameter angegeben werden, ein kurzes Beispiel ist (in Groovy steht `[:]` für eine leere Map):

```
BatixRecord[] recs = tmdKunden.findRecords([
  ["archived", 4, 1] // alle archivierten Kunden finden
], [:])
```

Ab Version 2.7.0 können auch mehrere Sortierungs-Kriterien angegeben werden. Dabei sind folgende Formate für `orderby` möglich (`desc` wird dann ignoriert):

```
BatixRecord[] recs = tmdKunden.findRecords([], [
  orderby: [
    "Firma", // aufsteigend
    ["Name"], // aufsteigend
    ["Vorname", true] // absteigend
  ]
])
```

Einen Datensatz filtern

Ab Version 2.7.0 ist es auch möglich nur den ersten Datensatz zu laden, anstatt alle:

```
BatixRecord rec = tmdKunden.findRecord([
  // siehe oben
], [
```

```
  // siehe oben
])
```

Rückgabewert ist entweder ein `BatixRecord` oder `null`.

Batix-Quelltext ausführen

Will man schnell einen String evaluieren, der Batix-Tags enthält, kann diese Methode benutzt werden:

```
String code = '<bx:math>1 + 1</bx:math>'
String result = code.evaluateAsBatixSource()
```

Der Standard Mime-Typ ist text/html, um einen anderen zu benutzen (manche Tags machen unterschiedliche Sachen, je nach Mime-Typ der Seite) kann dieser als Parameter mitgegeben werden: `evaluateAsBatixSource("text/plain")`.