bx:xml

Ab Version 2.6.8

Mit dem Tag `bx:xml` können XML Strukturen ausgelesen und einfach durchgegangen werden. Um schnell in tief verschachtelte Objekte zu gelangen, kann GPath genutzt werden.

XML laden

```
<bx:xml data="[1, 2, 3]">...</bx:xml>
<bx:xml data="clipboard:xml_data">...</bx:xml>
<bx:xml url="http://server/..." [encoding="utf-8"]>...</bx:xml>
```

Das XML-Dokument kann entweder direkt als Text übergeben werden (wobei der Umweg über ein Clipboard oder Attribut möglich ist), es kann aber auch eine URL angegeben werden, von der die Daten geladen werden (`encoding` gibt dabei an, mit welchem Zeichensatz die Daten eingelesen werden, Standard ist utf-8).

> Im URL-Modus sendet das XML-Tag nicht das Session-Cookie mit, da meist externe URLs aufgerufen werden und die SESSIONID nicht ausversehen preisgegeben werden soll.

Falls die Daten von einem internen, geschützten Menüpunkt geladen werden (auf den der aktuell eingeloggte Benutzer Zugriff hat), kann `<a href="http://docs.batix.info/display/DEVS/bx:jspinclude" target="_blank">bx:jspinclude</a>` verwendet werden - dieses schickt das Session-Cookie mit:

```
<bx:clipboard.cut
name="xml_data"><bx:jspinclude>/intern/api.xml</bx:jspinclude></bx:clipboard.cut>
<bx:xmln data="clipboard:xml_data">
  ...
</bx:xml>
```</p>

Ist in den XML-Daten ein Objekt kodiert, kann dieses mit geschachtelten xml-Tags durchgegangen werden; ein Array über gleichnamige Tags kann simuliert werden.

Eine fortgeschrittenere Möglichkeit des Daten-Ladens, kann über das Attribut `application` realisiert werden. Hier gibt man den Name eines Application-Attributes an, in dem

````
eine `Node` oder `NodeList` gespeichert ist. Dieses wird dann vom Tag benutzt und es entfällt das String-Parsen. Dies ist nützlich bei Caches, die man global in der Application vorhalten kann, und die sich nicht oft ändern. Ein entsprechendes Objekt kann z.B. mittels Groovys `XmlParser` erzeugt werden.

## XML durchgehen

Die XML-Struktur kann einfach via Tags durchgegangen werden:

```xml
<bx:xml data="..."> <!-- Root-Tag -->
  <bx:xml.Kunde> <!-- Sub-Tag -->
    Der Kunde heißt: <bx:xml.Vorname /> <bx:xml.Nachname />
    <bx:xml field="address_home">
      Er wohnt in: <bx:xml.Ort />
    </bx:xml>
  </bx:xml.Kunde>
</bx:xml>
```
````

Tag-Attribute können via @ angesprochen werden:

```
<!-- XML-Daten -->
...<age underage="false">18</age>...

<!-- Attribut age abfragen -->
...
<bx:xml.age@underage bool>Zutritt verboten!</bx:xml.age@underage>
...
```

Es ist jeweils die Angabe von `[encode=...](/books/cms-handbuch-entwickler/page/encodings-uebersicht)` möglich, um z.B. Ausgaben für URLs oder CSV zu encoden.

Das auszugebende Feld kann entweder im Tag-Titel (nach `bx:xml.`, also im Beispiel "Kunde") oder im Parameter `field=""` (oben z.B. "address_home") angegeben werden. Die Angabe via `field` ist nötig, falls der Feldname Sonderzeichen / Leerzeichen enthält.

Wird versucht auf ein Feld zuzugreifen, das nicht existiert, wird nichts ausgegeben. Es kann alternativ via `dummy` ein Alternativwert (als komplettes XML-Tag) angegeben werden, der stattdessen genommen wird.

```
<bx:clipboard.cut name="xmldummy">
  <object>
```

```
    <some></some>
    <item></item>
  </object>
</bx:clipboard.cut>
<bx:xml.missingObject dummy="clipboard:xmldummy">
  <bx:xml.some />
</bx:xml.missingObject>
```

Da XML-Tags (sowohl Root-Tags als auch Sub-Tags) mehrfach verschachtelt sein können und man manchmal Sachen von Über-Über-...-Elementen ausgeben will, kann man via Tag-Titel oder `name` die Tags benennen und via `base` diese referenzieren (der Parameter `name` hat beim Suchen des passenden Parent-Tags Vorrang vor dem Tag-Titel). Das entspricht der Funktionalität von `baseloop` bei Containern. Mittels `base="$"` kann das Root-Tag referenziert werden (das Tag, bei dem die Daten geladen wurden). Die Angabe von base kann immer zusätzlich zu allen anderen Parametern erfolgen (z.B. können `base` und `path` gleichzeitig benutzt werden).

```
<bx:xml data="..."> <!-- Root-Tag -->
  <bx:xml.Sub>
    <bx:xml.SubSub name="Kunde">
      <bx:xml.SubSubSub>
        <bx:xml.Feld base="Sub">
          <!-- selber Kontext, als wenn man direkt innerhalb von <bx:xml.Sub> wäre -->
        </bx:xml.Feld>
        <bx:xml.Feld base="Kunde">
          <!-- selber Kontext, als wenn man direkt innerhalb von <bx:xml.SubSub> wäre -->
        </bx:xml.Feld>
        <bx:xml.Feld base="$">
          <!-- selber Kontext, als wenn man direkt innerhalb des Root-Tags wäre -->
        </bx:xml.Feld>
      </bx:xml.SubSubSub>
    </bx:xml.SubSub>
  </bx:xml.Sub>
</bx:xml>
```

Je nach Datentyp muss ein entsprechender Parameter im bx:xml-Tag übergeben werden, da XML an sich keine Datentypen kennt (Schemas werden nicht unterstützt):

Null

Der Inhalt wird ausgeführt (oder nicht bei `not`), falls im XML das gesuchte Feld nicht vorhanden ist. Es wird hier auch der Spezialtyp `nil` einer XML-Schema-Instanz beachtet.

```
<! -- Feld in XML nicht enhalten oder als nil mit korrektem Namespace markiert -->
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <mynull xsi:nil="true" />
</root>

<bx:xml.Feld null>Feld war nicht vorhanden oder nil</bx:xml.Feld>
<bx:xml.Feld null not>Feld war vorhanden und nicht nil</bx:xml.Feld>
```

Boolean

Boolesche Werte können ähnlich wie Häkchenfelder im Container abgefragt werden. Eine kurze Version (geschlossenes Tag) mit `true` / `false` steht auch zur Verfügung.

```
<bx:xml.Condition bool [not]>
  ...
</bx:xml.Condition>

<bx:xml.Condition bool [true=""] [false=""] />
```

Um dem booleschen Wert `true` zu entsprechende, muss der Feldwert einer der folgenden Strings sein:

- true
- wahr
- yes
- y
- ja
- j
- on
- x
- 1
- 1.0

Number

Zahlen (mit und ohne Dezimalstellen) können via `num` angesprochen werden. Das Tag erwartet als Dezimaltrenner einen Punkt. Die Angabe von `comma` bewirkt, dass vorher ein Komma im Feldwert durch einen Punkt ersetzt wird.

ausgeben / formatieren

Zahlen können formatiert ausgegeben werden. Ist kein `pattern` angegeben, werden Ganzzahlen *ohne* und Kommazahlen *mit* Nachkommastellen ausgegeben. Der Tausender- und Dezimal-Trenner

kann mittels `gs` bzw. `ds` spezifiziert werden. Der Rundungsmodus ist standardmäßig HALF_UP (normales kaufmännisches Runden) und kann via `rounding` geändert werden (siehe Java-Docs für mögliche Werte).).

```
<bx:xml.Zahl num [comma] [pattern=""] [locale=""] [gs=""] [ds=""] [rounding=""] />
```

`locale` sollte immer angegeben werden, da sonst die Standardeinstellung vom Server genutzt wird, die sich aber nach Updates oder Umzügen ändern kann.

vergleichen

Folgende Vergleiche sind möglich:

Parameter	ausgeführter Vergleich
equals	==
gt	>
gte	>=
lt	<
lte	<=

Das Ergebnis kann jeweils mit `not` umgekehrt werden. Für alle Vergleiche ist auch eine verkürzte Variante mit `true` / `false` möglich (hier wird der jeweilige Wert von `true` oder `false` ausgegeben).

```
<bx:xml.Zahl num [comma] [equals=""] [gt=""] [gte=""] [lt=""] [lte=""] [not]>
</bx:xml.Zahl>

<bx:xml.Zahl num [comma] [equals=""] ... [true=""] [false=""] />
```

Date

Über den Tag-Parameter `date` wird der vorhande String-Wert im Feld als Datum interpretiert.

Eingabeformat

Ist lediglich `date` ohne weiteren Wert im Tag angegeben, werden einige Standard-Patterns unterstützt:

```
<bx:xml.mydate date />
```

Es wird folgende Liste in dieser Reihenfolge durchprobiert:

- `dd.MM.yy HH:mm:ss`
- `dd.MM.yy HH:mm`
- `dd.MM.yy HH.mm`
- `dd.MM.yy`
- `yyyy-MM-dd'T'HH:mm:ssZ` (ISO 8601)

Alternativ kann ein eigenes Pattern angegeben werden:

```
<bx:xml.mydate date="dd/MM/yyyy" />
```

Es muss nicht der gesamte String matchen. Möchte man z.B. aus dem Wert "20.08.2018 um 08:50" nur das Datum ziehen, genügt das Pattern `dd.MM.yyyy`.

Ausgabeformat

Standardmäßig wird das geparste Datum anhand des Patterns `dd.MM.yyyy HH:mm:ss` ausgegeben. Es kann auch ein eigenes Pattern angegeben werden:

```
<bx:xml.mydate date pattern="dd.MM. HH:mm" />
```

Werden z.B. Tages- oder Monatsnamen ausgegeben, wird über `locale` die gewünschte Sprache eingestellt:

```
<bx:xml.mydate date pattern="EEEE, d. MMMM" locale="de" />
```

Datum/Zeit modifizieren

Durch die Angabe von verschiedenen Parametern können alle Teile des Ausgangsdatums angepasst werden. Es kann eine Zahl angegeben werden, um den entsprechenden Wert absolut zu setzen. Dabei kann dieser Zahl ein Plus (`+`) oder Minus (`-`) vorangestellt werden, um die Modifikation stattdessen relativ durchzuführen. Die möglichen Parameter sind (werden auch in dieser Reihenfolge angewandt):

- year
- month (im Bereich 1 - 12)
- day
- weekday (s.u.)
- hour
- minute
- second

Etwaige Überschreitungen (z.B. +15 Monate) werden entsprechend weiter gezählt (also +1 Jahr und 3 Monate).

Beispiel (Tag auf den 15. Februar setzen, zwei Stunden dazu zählen, zehn Jahre abziehen):

```
<bx:xml.mydate date day="15" month="2" hour="+2" year="-10" />
```

Mögliche Werte beim Wochentag sind dabei folgende Werte (keine Zahlen erlaubt):

- mo / di / mi / do / fr / sa / so
- mon / tue / wed / thu / fri / sat / sun

Es wird auf den entsprechenden Wochentag in der gleichen Woche gewechselt. Auch hier gibt es die Möglichkeit ein Plus oder Minus voranzustellen, dann wird entsprechend auf den Tag in der nächsten bzw. vorherigen Woche gewechselt.

Trifft der angegebene Wochentag bereits auf das Ausgangsdatum zu, wird das Datum nicht geändert. Dieses Verhalten kann durch Angabe eines doppelten Plus (`++`) oder Minus (`--`) überschrieben werden (es wird dann auch gewechselt, falls der Wochentag schon zutrifft).

Zeitliche Abfrage

Eine weitere Funktion ist die Abfrage, ob das Ausgangsdatum (bzw. das modifizierte Datum) vor oder nach dem jetzigen Zeitpunkt (Zeit auf dem Server) bzw. heute liegt. Der Unterschied ist, dass beim Vergleich mit **heute** die Uhrzeit nicht beachtet wird, sondern nur der Datumsteil.

Die folgenden Abfragen sind möglich:

- before-now
- after-now
- before-today
- after-today
- today (ist am heutigen Tag)

Das Tag kann dabei entweder offen (optional noch mit `not`) oder geschlossen mit den `true` und/oder `false` Parametern genutzt werden. Beispiele:

```
<bx:xml.mydate date before-now>in der Vergangenheit</bx:xml.mydate>
<bx:xml.mydate date after-now not>auch in der Vergangenheit oder genau jetzt</bx:xml.mydate>

<bx:xml.mydate date before-today true="vor heute" />
<bx:xml.mydate date after-today false="vor heute oder heute" />

<bx:xml.mydate date today true="am heutigen Tag" false="nicht heute" />
```

String

ausgeben

Im einfachsten Fall wird der String ausgegeben und ggf. automatisch encoded (siehe Encodings für andere Encoding-Möglichkeiten).

```
<bx:xml.Text />
```

vergleichen

Strings können auch verglichen werden. Die Angabe von `ignoreCase` bewirkt, dass Groß-/Kleinschreibung nicht beachtet wird, `not` kehrt das Ergebnis um.

Parameter	Vergleich
matches	Vergleich via RegEx (regulärem Ausdruck), String muss diesem komplett entsprechen
equals	Gleichheit
contains	der String muss den Wert das Parameters beinhalten

Auch hier ist eine verkürzte Variante mit `true` und `false` möglich.

```
<bx:xml.Text [matches=""] [equals=""] [contains=""] [ignoreCase] [not]>
</bx:xml.Text>

<bx:xml.Text [matches=""] [equals=""] [contains=""] [ignoreCase] [true=""] [false=""] />
```

BatixRecord

Steht in den XML-Daten im Feld eine Batix-ID, so kann hier auch ein Container angebunden werden. Das Tag verhält sich dann wie `bx:record`, es können innerhalb die normalen Container-Tags wie `bx:recordfield` oder `bx:recorddata` benutzt werden. Auch der Zugriff von außerhalb des Tags mit z.B. `bx:recorddata.nav` ist möglich.

Der Datensatz wird anhand des Feldes ID rausgesucht. Es kann aber auch via `linkfield` auf ein anderes Feld im Datensatz verwiesen werden (Text oder Einzelverknüpfung), hierbei wird der erste gefundene Datensatz genommen.

Wird kein Datensatz mit entsprechender ID (oder Wert im Feld) gefunden, wird nichts ausgegeben. Die Angabe von `dummy` bewirkt, dass stattdessen ein leerer Datensatz vorgehalten wird (der Tag-Inhalt wird ausgeführt, innere `bx:recordfield`-Tags etc. geben aber nichts aus, wie bei `[bx:record](/books/cms-handbuch-entwickler/page/bx-record)`).

```
<bx:xml.IDFeld pool="Container" [linkfield=""] [dummy]>
  <bx:recordfield... />
  <bx:recorddata.if... />
</bx:xml.IDFeld >
```

```
<bx:recorddata.nav object="IDFeld">
  <bx:recorddata.total />
</bx:recorddata.nav>
```

Array

Gleichnamige XML-Tags können mittels `array` in einer Schleife durchlaufen werden.

Ein Array funktioniert analog zu z.B. `bx:containerfilter`, für jedes Element wird der Tag-Inhalt einmal ausgeführt. Das Start-Element kann dabei mittels `index` (0-basiert) und die maximale Anzahl Durchläufe via `max` geregelt werden.

Um das aktuelle Element im Durchlauf auszugeben wird einfach `<bx:xml />` verwendet. Je nach Datentyp (in Arrays können auch verschiedene Datentypen vorhanden sein) können hier die entsprechenden Parameter und deren Funktionen verwendet werden (z.B. formatierte Ausgabe bei Zahlen).

```
<!-- XML Teil -->
<Liste>10</Liste>
<Liste>20</Liste>
<Liste>30</Liste>

<!-- durchgehen -->
<bx:xml.Liste array [index="20"] [max="10"]>
  <bx:xml />
</bx:xml.Liste>
```

Es können die meißten Funktionen von `[bx:recorddata](/books/cms-handbuch-entwickler/page/bx-recorddata)` verwendet werden:

```
<bx:xml.Liste array>
  ...
</bx:xml.Liste>
<bx:recorddata.nav object="Liste">
  Gesamt: <bx:recorddata.total/><br>
  <bx:recorddata.navlist max="5">
    ...
```

Ferner gibt es noch folgende Hilfsmethoden:

```
<bx:xml.Liste array>
  <bx:xml {first|last} [not]>...</bx:xml> <!-- Inhalt nur ausführen, wenn es das erste oder
letzte Element ist -->
  <bx:xml {first|last} [true=...] [false=...] /> <!-- dito, nur verkürzte true/false
Schreibweise -->
  <bx:xml index [add="5"] /> <!-- aktuellen Durchlauf-Index ausgeben, ggf. etwas dazu addieren
-->
  <bx:xml total /> <!-- Gesamtanzahl der Elemente ausgeben -->
  <bx:xml empty [not]>...</bx:xml> <!-- ausführen, falls das Array leer ist -->
  <bx:xml empty [true=...] [false=...] /> <!-- dito, nur mit true/false -->
  <bx:xml cols=...> <!-- funktioniert wie bx:recorddata.cols -->
</bx:xml.Liste>
```

Für mehr Informationen zu `cols` siehe die entsprechende Passage bei `bx:recorddata`.

Datensätze

Befinden sich im Array Strings, welche Batix-IDs sind, kann mittels `pool` eine Schleife über Datensätze (wie z.B. `bx:containerfilter`) aufgemacht werden. Falls die ID in einem anderen Feld steht, kann dies wieder via `linkfield` referenziert werden. Innerhalb des Xml-Tags sind alle normalen Datensatz-Tags verfügbar. `dummy` verhält sich im Falle eines leeren Arrays wie ein leerer Datensatz (ansonsten wird der Inhalt nicht ausgeführt).

Falls im Array (zusätzlich) nicht-Strings sind, wird im Log eine Warnung ausgegeben.

```
<bx:xml.Kunden array pool="Shop_Kunden" [linkfield=""] [dummy]>
  <bx:recordfield... />
  <bx:recorddata.if... />
</bx:xml.Kunden>
```

Object

Um in ein XML-Objekt tiefer einzusteigen, wird das Xml-Tag (analog zu `bx:recordfield` bei Verknüpfungen) geschachtelt.

```
<bx:xml data="...">
  <bx:xml.Kunde>
    <bx:xml.Name />
  </bx:xml.Kunde>
</bx:xml>
```

Namespaces

Namespaces werden ebenfalls grundlegend unterstützt (es kann nach dem Namespace-Präfix gesucht werden, der Namespace-Identifier (meist eine URI) wird ignoriert). Hierbei muss für den Feldname allerdings `field` benutzt werden, da der Doppelpunkt sonst Probleme macht:

```
<!-- XML Teil -->
<nstest>Namespace 0</nstest>
<ns1:nstest>Namespace 1</ns1:nstest>
<ns2:nstest>Namespace 2</ns2:nstest>

ohne: <bx:xml field="nstest" />
1: <bx:xml field="ns1:nstest" />
2: <bx:xml field="ns2:nstest" />
alle: <bx:xmL field="*:nstest" array>...</bx:xmL> <!-- Elemente ohne expliziten Namespace sind
nicht enthalten -->
```

GPath

Mittels GPath-Expressions können relativ einfach verschachtelte Strukturen angesprochen werden. Das erspart lästiges Verschachteln von Xml-Tags.

Operator	Bedeutung
`*`	children
`**`	depth-first

Beispiele:

```
<bx:xml path="mystring" />
<bx:xml path="mystring@name" />
<bx:xml path="mystring.@name" />
<bx:xml path="myobj.name" />
<bx:xml path="myobj.age@underage" />
<bx:xml path="myobj.age.@underage" />
<bx:xml path="**.@title" />
<bx:Xml path="**.@title" array>[<bx:xml />]</bx:Xml>
<bx:Xml path="*.@title" array>[<bx:xml />]</bx:Xml>
<bx:Xml path="myobj.*" array>[<bx:xml />]</bx:Xml>
```