Tag

Das Framework kann durch Plugins um Batix-Tags (`<bx:tagname>`) erweitert werden, welche dann in normalen Quelltexten wie Komplettseiten oder Textbausteinen verwendet werden können.

```
fun registerTag(tagInfo: TagInfo)
```

Es gibt zwei Arten von Tags: Frontend- und Backend-Tags. Der Unterschied besteht darin, dass sich Backend-Tags in der Verwaltung darstellen oder dort konfiguriert werden können (wie z. B. `<bx:text>` oder `<bx:containerfilter>`), Frontend-Tags hingegen können dies nicht.

Frontend-Tags

Frontend-Tags werden mithilfe von `TagInfo.frontend` registriert und müssen von `com.batix.plugins.PluginFrontendTag` abgeleitet sein. Es reicht im einfachsten Fall, die Methode `addFrontendSourceText(StringBuffer)` zu überschreiben. An den `StringBuffer` hängt man die Ausgabe an und gibt diesen am Ende wieder zurück.

```
registerTag(TagInfo.frontend("tagname", MyFrontendTag::class.java, null))
```

> **Injection Vulnerability**
> Es müssen (HTML-)Steuerzeichen passend encoded werden, um Injectionlücken zu vermeiden.

Es empfiehlt sich daher die Methode `writeEncodedOutput(String, StringBuffer)` zu benutzen. Dieser übergibt man als ersten Parameter den gewünschten (uncodierten) Ausgabetext und reicht als zweiten Parameter den `StringBuffer` durch, den man übergeben bekommen hat. Die Methode sorgt automatisch dafür, dass entsprechend der aktuellen Frontendseite das passende Encoding (z. B. htmlencode bei HTML-Seiten) gewählt wird. Außerdem unterstützt das Tag damit automatisch den Parameter `encode` (also z. B. `<bx:tagname encode="plain" />`).

Backend-Tags

Backend-Tags werden mithilfe von `TagInfo.backend` registriert und müssen von `com.batix.plugins.PluginBackendTag` abgeleitet sein. Hier müssen neben `addFrontendSourceText(StringBuffer)` noch die Methoden `getDataTable()` und

`addAdminSourceText(StringBuffer)` überschrieben werden.

```
registerTag(TagInfo.backend("tagname", MyBackendTag::class.java, null))
```

Der erste Parameter (`"tagname"`) ist dabei der Name des Tags, wie er dann auch hinter `<bx:` verwendet wird. Als zweiter Parameter wird die implementierende Klasse übergeben. Der letzte Parameter ist ein optionales String-Array, falls das Tag nur für bestimmte Projekte (anhand webdir) zur Verfügung stehen soll.

Ist das Tag offen verwendbar (also z. B. `<bx:tagname>etwas Inhalt...</bx:tagname>`), kann der Inhalt (*Body*) mittels `computeBody()` ausgeführt und das Ergebnis ausgelesen werden. Eventuelle Batix-Tags im Body werden damit auch evaluiert.

Parameter

Einem Tag können im Quelltext Parameter übergeben werden.

```
<bx:tagname mode="short" maxlen="5" pretty />
```

Diese Parameter können mit den `get*Parameter(key: String)`-Methoden ausgelesen werden – `key` ist der Name des Parameters. Mit `containsParameter(key: String)` kann festgestellt werden, ob ein Parameter angegeben wurde oder nicht.

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

fun getIntParameter(key: String): Integer
fun getIntParameter(key: String, defaultValue: Integer): Integer

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

Backend-Tag Speicherung

Backend-Tags stellen ein Eingabeelement für Redakteure oder Admins bereit. Dafür muss im Quellcode ein sogenannter Titel am Tag vergeben werden. Bei `<bx:tagname.Anrede />` ist der Titel beispielsweise "Anrede". Dieser wird dem Benutzer im Backend angezeigt.

Das bedeutet, dass die vom Benutzer getätigten Eingaben in der Datenbank gespeichert werden müssen. Die Methode `getDataTable()` teilt dem Framework mit, in welcher Tabelle die Daten zu speichern sind. Für kurze, einzeilige Texte ist das "EZT", für längere Texte "MZT". Der

entsprechende Wert ist von `getDataTable()` einfach als String zurückzugeben.

Die Methode `addAdminSourceText(StringBuffer)` ist für das Rendern des entsprechenden Eingabefeldes zuständig. Für kurze Texte kann das ein `<input type="text">` sein. Der Name des Inputs muss als Prefix die Tabelle (gleicher Wert wie bei `getDataTable()`), einen Punkt als Separator und als Suffix den Titel des Tags enthalten (dieser ist als Feld `titel` verfügbar). Er muss also beispielsweise "EZT.Anrede" lauten. Ist bereits ein Wert gespeichert, ist das Feld `dataId` gefüllt. Dessen Wert muss dann noch, inklusive einem weiteren Punkt, an den Name angehangen werden.

> **Vorgefertigte Methoden**
> Als Best-Practice empfiehlt sich die Verwendung der Methoden `appendAdminHeadline(StringBuffer)` und `appendPublishStatus(StringBuffer)`, um eine konsistente Ausgabe des Titels und des Veröffentlichungsstatus zu erreichen.

Der Aufruf `getData("INHALT")` gibt den aktuell in der Datenbank gespeicherten Wert zurück. Dieser kann dann zur Ausgabe im Frontend sowie zur Vorbefüllung des Eingabefelds im Backend verwendet werden.

Beispiel

Frontend-Tag

Das Tag wird in der Plugin-Hauptklasse mithilfe von `TagInfo.frontend` registriert.

```
override fun load() {
  registerTag(TagInfo.frontend("unixtime", UnixTimeTag::class.java, null))
}
```

Dieses Beispiel-Tag gibt den aktuellen Epoch-Timestamp aus.

```
import com.batix.plugins.PluginFrontendTag
import com.batix.tags.BatixTagData
import java.time.Instant

class UnixTimeTag(data: BatixTagData?) : PluginFrontendTag(data) {
  override fun addFrontendSourceText(sb: StringBuffer): StringBuffer {
    val time = Instant.now().epochSecond
    writeEncodedOutput(time.toString(), sb)
    return sb
```

```
  }
}
```

Der Aufruf im Quellcode ist minimal.

```
<bx:unixtime />
```

Die Ausgabe im Frontend ist dann z. B. *1584620787*.

Backend-Tag

In der Plugin-Hauptklasse wird das Tag mit `TagInfo.backend` registriert.

```
override fun load() {
  registerTag(TagInfo.backend("formattedtime", FormattedTimeTag::class.java, null))
}
```

Die Tag-Klasse implementiert die Frontend-Logik sowie die Anzeige des Eingabeelements im Backend.

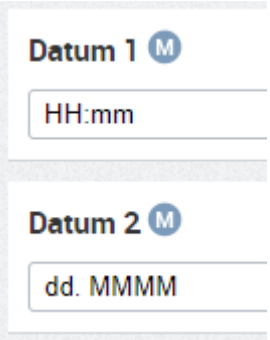
```
import com.batix.Tools
import com.batix.plugins.PluginBackendTag
import com.batix.tags.BatixTagData
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter
import java.util.*

class FormattedTimeTag(data: BatixTagData?) : PluginBackendTag(data) {
  override fun addFrontendSourceText(sb: StringBuffer): StringBuffer {
    // Backendeingabe lesen
    val pattern = getData("INHALT") as String? ?: "dd.MM.yyyy HH:mm:ss"

    // Parameter aus Quelltext lesen
    val locale = Locale.forLanguageTag(getStringParameter("locale", "de-DE"))

    val now = LocalDateTime.now()
    val formatter = DateTimeFormatter.ofPattern(pattern, locale)
    writeEncodedOutput(formatter.format(now), sb)
    return sb
  }
```

```
  override fun getDataTable(): String {
    return "EZT"
  }

  override fun addAdminSourceText(sb: StringBuffer): StringBuffer {
    val inhalt = getData("INHALT") as String? ?: ""

    appendAdminHeadline(sb)
    appendPublishStatus(sb)

    var inputName = "$dataTable.$titel"
    if (dataId != null && dataId != "del") {
      inputName += ".$dataId"
    }

    sb.append("""<input type="text" name="${Tools.htmlEncode(inputName)}" """)
    sb.append("""value="${Tools.htmlEncode(inhalt)}">""")

    return sb
  }
}
```

Im Quelltext wird das Tag dann inklusive Titel verwendet.

```
<bx:formattedtime.Datum_1 />

<bx:formattedtime.Datum_2 locale="en-US" />
```

In der Verwaltung können Eingaben getätigt werden.



Im Frontend wird dann z. B. folgender Text ausgegeben:

13:28

19. March