JART

Development

- **JART Syntax**
- **JART XSL Extension**

Developer-Apps

DB-App-Builder

Der DB-App-Builder ist ein grafisches Interface mit dessen Hilfe SQL Statements erstellt werden können. Eine fertige Selektion wird in einem XML File gespeichert. und kann anschließend mit der Function run-selection im Jart Code aufgerufen werden.

```
<art:call function="run-selection" selection="path/to/selection.xml"></art:call>
```

Allgemeines

Selektionen im DB-App-Builder sind bereits gegen SQL Injections abgesichert, d.h. es brauchen keine zusätzlichen Vorkehrungen getroffen werden. Um den DB-App-Builder zu verwenden muss folgende Datei verwendet werden.

```
<art:include href="/prj3/jart-tools/resources/developer-apps/db-app-builder/includes/dba
```

Die Beziehungen zwischen den einzelnen Tabellen werden vom DB-App-Builder aus dem dbcon.xml File gelesen (meistens in prj3/projekt/resources/dbcon.xml zu finden). Falls keine Ajax DB im aktuellen Projekt verwendet wird müssen die Relationen erst mit dem Relationseditor gesetzt werden, da diese die Einträge im dbon.xml generiert.

Der DB-App-Builder kann mit dem Developer Tools Icon auf der Startseite aufgerufen werden. (Schraubenschlüssel). Wenn man sich im DB-App-Builder Interface befindet gibt es die Möglichkeit einen Folder anzulegen bzw. zu wählen und darin eine Selektion anzulegen bzw. auszuwählen. Im folgenden werden die Möglichkeiten beschrieben um eine Selektion zu erstellen.

Eine einfach Selektion

Wenn man sich im Zielfolder für die Speicherung der Selektion befindet, muss in der Selectbox "selection" der eintrag new gewählt werden. Zuerst muss man sich nur auf die rechte Seite des Bildschirms konzentrieren, hier befinden sich die Eigenschaften. Beginnt man bei der Bearbeitung mit der Wahl der richtigen Tabelle wird der Name der Tabelle automatisch auch als Selektionsname und Nodename gesetzt, diese Felder können natürlich auch manuelle bearbeitet werden. Falls eine Verbindung mit anderen Tabellen besteht werden diese im Feld Subtables zur Auswahl angeboten, darauf wird später eingegangen. Die Eigenschaft Fields erstellt eine Liste aller Felder aus der DB. Mittels * wird alles selektiert, mit *.+ wird alles außer _history und jartdb_deleted gewählt, *.- bewirkt das Gegenteil. Während die Felder angeklickt werden ist in der linken Hälfte des Interface die getätigte Auswahl zu

sehen. Die getroffenen Einstellungen reichen bereits aus um eine einfache Selektion zu erstellen, die Selektion muss nur noch mittels Save gespeichert werden.

Globalisierung von Paketen

Globalisierung

1. Content-Designer oben auf "go" (vom Localhost - also leer)
2. bei Versionscheck: Paket publizieren -->Globalisierung des Pakets abgeshclossen
3. "Jetzt aktualisieren" ->aktualisiert das Paket im aktuellen Projekt

Globale Library

Struktur: standard, global, database, special

Pfad ändern

Im Content-Designer bei den Paketen beim Pfad auf "ändern" klicken ->Pfad ändern ->dann auf "go" und beim Paket "Paket publizieren"

JART Syntax

JART Nodes

die globalen Attribute sind für alle JART Nodes gültig

Neu

- art : script

Logik und Steuerung

- art : variable
- art : date
- art : block
- art : call

- art : with-node
- art : if
- art : choose
  - art : when
  - art : otherwise
- art : while
- art : for
- art : foreach
- art : check-illegal
- art : wait
- art : synced

- art : include

- art : plain
- art : element
- art : attribute
- art : text
- art : comment
- art : xml

- art : move-node
- art : delete-node
- art : rename
- art : sort

- art : transform
- art : namespace
- art : move-ns-to-normal
- art : push
- art : send-mail

Datenbank

- art : jdbc
- art : db-update
- art : db-select
- art : db-execute
- art : db-all-columns
- art : db-all-tables

Dateien

- art : get-xml
- art : save-node
- art : file
- art : read-doc
- art : xls
- art : multi-file

Andere Datenquellen

- art : lucene4
- art : lucene
- art : ldap
- art : soap-client
- art : get-post-data
- art : send-post-data

- art : http-client
- art : html2xml

- art : session-data
- art : session
- art : cookie
- art : cache

- art : zip

Bilder und Rendering

- art : image
- art : svg
- art : pdf
- art : pdf-adv
- art : pdf2image

Sonstige

- art : debug
- art : redirect

- art : python
- art : exec

- art : glossary
- art : add-header
- art : all-http-header
- art : security

- art : sysinfo
- art : system-info
- art : math

Spezial

- art : script
- art : picasa
- art : saverpay
- art : ignore-ssl-verify
- art : recaptcha
- art : ntlm

OUT OF DATE

- art : fx-send
- art : lucene-ext
- art : dbq-update
- art : multi-form
- art : minify

Globale Attribute

Attribute

@ key

@ on-error

Beispiele

art : attribute

jart.handlers.basic.JArtAttribute

Beschreibung

Mittels art:attribute kann ein Attribut bei einem XML-Knoten erstellt oder gelöscht werden. Weiters können alle zur Laufzeit vorhandenen Variablen mittels dem "dump-var"-Attribut als Attribute eines XML-Knotens ausgegeben werden.

Attribute

Globale Attribute

@ name *(erforderlich)*

Das "name"-Attribut definiert den Namen des neuen Attributs.

@ value *(optional)*

Mit dem "value"-Attribut kann dem neu erzeugten Attribut ein Wert zugeweisen werden.

@ remove *(optional)*

Wird das "remove"-Attribut auf "yes" gesetzt, wird das Attribut das im "name"-Attribut angegeben ist, gelöscht.

@ target *(optional)*

Mittels des "target"-Attributs kann über XPath ein XML-Knoten ausgewählt werden in dem das Attribut erzeugt oder gelöscht wird. Ist "target" nicht angegeben wird das Attribut im aktuellen XML-Knoten erzeugt.

@ prefix *(optional)*

Das "prefix"-Attribut definiert den Namespace Prefix welcher aber über art:namespace gesetzt werden muss.

@ dump-vars *(optional)*

Wird das "dump-vars"-Attribut auf "yes" gesetzt, werden alle zur Laufzeit vorhandenen Variablen als Attribute in den XML-Knoten geschrieben.

Beispiele

```
<art:plain name="data">
   <art:attribute name="attr1" value="Hello Universe!" />
   <art:plain name="node">
      <art:attribute name="attr2" value="Hello World!" />
      <art:attribute target=".." remove="yes" name="attr1" />
   </art:plain>
</art:plain>
```

```
Ergebnis (XML):
<data>
   <node attr2="Hello World!" />
</data>
```

Dump-Vars

```
<art:plain name="data">
   <art:variable name="foo1" value="Hello World!" />
   <art:variable name="foo2" value="Hello Universe!" />
   <art:attribute dump-vars="yes" />
</art:plain>

Ergebnis (XML):
<data foo2="Hello Universe!" foo1="Hello World!" />
```

art : block

jart.handlers.control.Block

Beschreibung

art:block ist eine Klasse die einerseits für die übersichtliche Anordnung von Code-Blöcken dient die optional mit key-Verhalten versehen werden können und andererseits um Funktions-Blöcke zu definieren die über art:call aufgerufen werden können.

Attribute

Globale Attribute

@function (optional)

Im "function"-Attribut kann ein Funktionsname vergeben werden, welcher über [../art_call art:call aufgerufen werden kann. Ist ein Funktionsname mehr als einmal in Verwendung werden alle Blöcke mit diesem Namen beim Aufruf der Funktion abgearbeitet.

@ comment (optional)

Das "comment"-Attribut dient dazu, Blöcke mit Kommentaren zu versehen um die Übersichtlichkeit im Code zu steigern.

Beispiele

```
<art:plain name="data">
   <art:block function="recurse" comment="i am a function">
      <art:plain name="node">
         <art:if test="count(ancestor::node) < 3">
            <art:call function="recurse" />
         </art:if>
      </art:plain>
   </art:block>
</art:plain>

Ergebnis (XML):
<data>
   <node>
```

```
      <node>
         <node>
            <node />
         </node>
      </node>
   </node>
</data>
```

Einfacher block ohne eigene Funktionalität ausser den Globale Attributen

```
<art:block key="foo">
    ....
</art:block>
```

Einfacher block ohne eigene Funktionalität nur zur Strukturierung verwendet

```
<art:block comment="was passiert hier">
    ....
</art:block>
```

Definition einer Funktion die durch den key "never" nicht im normale Ablauf ausgeführt wird und nur mit dem entsprechenden art:call ausgeführt werden kann

```
<art:block key="never" function="do_something">
    ....
</art:block>
```

art : check-illegal

jart.handlers.basic.CheckIlleagleParams

art:check-illegal

Attribute

Globale Attribute

@ values

Beispiele

art : choose

jart.handlers.control.Choose

Beschreibung

Der art:choose Knoten wird verwendet um eine Serie von Abfragen einzuleiten. Ein art:choose Knoten besitzt keine Attribute (abgesehen von optionalen globalen Attributen wie "key") und muss mindestens einen art:when Knoten enthalten. Es können beliebig viele art:when Knoten und optional maximal ein art:otherwise Knoten in einem art:choose Knoten angelegt werden.

art : comment

jart.handlers.basic.Comment

Beschreibung

art:comment ist eine Klasse um Kommentare im XML zu erzeugen.

Attribute

Globale Attribute

@ value

Hier wird der Kommentartext definiert.

@ target

Beispiele

```
<art:comment value="This is a comment" />

Ergebnis (XML):
<!-- This is a comment -->
```

art : date

jart.handlers.data.JArtDate

art:date

Beschreibung

art:date ist eine Klasse um ein Datum regionalabhängig zu erzeugen, formatieren oder zu parsen. Es ist möglich ein Datum in einem beliebigen Format zu erzeugen, ein vorhandenes Datum in Text zu formatieren und umgekehrt Text in ein Datum zu parsen.

Datum- und Zeitformate werden von benutzerdefinierten Mustern festgelegt. Innerhalb dieser Muster werden nicht in Anführungszeichen gesetzte Buchstaben (von A bis Z und von a bis z) als Muster-Zeichen interpretiert, die die Bestandteile eines Datum- oder Zeitformats vertreten.

Buchstabe	Datum- oder Zeitkomponente	Beispiele
G	Era designator	AD
y	Year	1996; 96
M	Month in year	July; Jul; 07
w	Week in year	27
W	Week in month	2
D	Day in year	189

d	Day in month	10
F	Day of week in month	2
E	Day in week	Tuesday; Tue
a	Am/pm marker	PM
H	Hour in day (0-23)	0
k	Hour in day (1-24)	24
K	Hour in am/pm (0-11)	0
h	Hour in am/pm (1-12)	12
m	Minute in hour	30
s	Second in minute	55
S	Millisecond	978
z	Time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	-0800

Diese Musterzeichen werden gewöhnlich wiederholt bis ihre Anzahl die exakte Darstellung widerspiegelt. Nicht reservierte Zeichen werden auch ohne Anführungszeichen wieder ausgegeben.

Muster	**Ergebnis**
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, "yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM
"hh 'o"clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2001-07-04T12:08:56.235-0700

Attribute

Globale Attribute

@ name (erforderlich)

Eindeutiger Bezeichner des Datumsobjekts

@ format (optional)

Das "format"-Attribut legt das Format des erzeugten Datumsobjekts fest.

Beispiel:

```
yyyy-MM-dd
```

@ input-date (optional)

Bei Übergabe eines vorhandenen Datums mittels des "input-date"-Attributes wird ein Datumobjekt anhand des übergebenen Datums erzeugt.

Beispiel:

```
dd.MM.yyyy HH:mm:ss
```

@ input-format (optional)

Im "input-format"-Attribut wird festgelegt, in welchem Format sich ein über das "input-date"-Attribut übergebenen Datums befindet.

Beispiel:

```
yyyyMMddHHmm
```

@ locale (optional)

Legt fest, auf Basis welcher Region das Datumsobjekt formatiert wird.

Mögliche Werte:

```
de, en, fr, it, jp
```

@ add (optional - benötigt field)

Mittels des "add"-Attributs ist es möglich Einheiten zu einem Datumsobjekt zu addieren bzw. abzuziehen. Welche Einheit angesprochen wird, wird über das "field"-Attribut definiert.

@ field (optional - benötigt add)

Über das "field"-Attribut wird festgelegt, welche Einheit über das "add"-Attribut angesprochen wird.

Mögliche Werte:

```
YEAR, MONTH, DAY, HOUR, MIN, SEC
```

@ number (optional)

Bei Verwendung des "number"-Attributes wird ein aktueller Timestamp in Millisekunden erzeugt und durch den Wert des "number"-Attributes dividert. Ein Wert von 1000 erzeugt somit einen Unix-Timestamp.

@ diff-to (optional)

Bei Übergabe eines Datums an das "diff-to"-Attribut wird ein Datumsobjekt erzeugt, welches den Unterschied beider Daten in Millisekunden enthält. Gerechnet wird: "diff-to" minus "input-date"

Beispiele

```
<art:date name="unix-timestamp" number="1000" />
<art:debug message="{$unix-timestamp}" />

<!--
  debug-info: 1204034115
-->


<art:date name="today" format="dd.MM.yyyy HH:mm" />
<art:debug message="{$today}" />

<!--
```

```
debug-info: 26.02.2008 15:05
-->


<art:date name="yesterday" format="EEEE" add="-1" field="DAY" />
<art:debug message="{$yesterday}" />

<!--
debug-info: Monday
-->


<art:date name="yesterday_de" format="EEEE" add="-1" field="DAY" locale="de" />
<art:debug message="{$yesterday_de}" />

<!--
debug-info: Montag
-->


<art:date diff-to="{$today}" input-format="dd.MM.yyyy HH:mm" input-date="01.01.1982 18:3(
<art:debug message="{$diff}" />

<!--
  debug-info: 825281280000
-->
```

art : file

jart.handlers.data.JArtFile

art:file

Ist für alle Datei und Ordner Operationen zuständig die nicht direkt mit XML Nodes zu tun haben

Attribute

Globale Attribute

@ action

Bestimmt die auszuführende Operation

Wert	Funktion
open	Öffnet eine Datei und gibt deren Inhalt wahlweise in der XML Struktur oder in einer Variablen über @variable angegeben aus

Beispiele

art : for

jart.handlers.control.For

Beschreibung

art:for dient als Kontrollstruktur, mit der man eine Gruppe von Anweisungen mit einer bestimmten Anzahl von Wiederholungen ausführen kann. Die aktuelle Position in der for-Schleife wird in der Variable $param gespeichert.

Attribute

Globale Attribute

@from (erforderlich)

Definiert den Startwert der for-Schleife.

@increment (erforderlich)

Definiert die Schrittweite mit der der Startwert verändert wird.

@to (erforderlich)

Definiert den Endwert der for-Schleife.

@param (optional)

Um die aktuelle Position nicht in der Variable $param zu speichern, kann hier ein anderer Variablenname gewählt werden. Dies macht vor allem bei verschachtelten for-Schleifen Sinn.

Beispiele

```
<art:plain name="data">
   <art:for param="x" from="1" increment="1" to="2">
      <art:plain name="node_{$x}">
         <art:for from="1" increment="1" to="2">
            <art:plain name="subnode_{$param}">
               <art:attribute name="parent-param" value="{$x}" />
            </art:plain>
         </art:for>
      </art:plain>
   </art:for>
</art:plain>

Ergebnis (XML):
<data>
   <node_1>
      <subnode_1 parent-param="1" />
      <subnode_2 parent-param="1" />
   </node_1>
   <node_2>
      <subnode_1 parent-param="2" />
      <subnode_2 parent-param="2" />
   </node_2>
</data>
```

art : foreach

jart.handlers.control.ForEach

Beschreibung

Die art:foreach Klasse kann verwendet werden um einen Block von Anweisungen auszuführen. Die Anzahl der Wiederholungen wird mittels eines Strings definiert, der nach verschiedenen zu definierenden Bedingungen gesplittet wird.

Attribute

Globale Attribute

@values (erforderlich)

Im "values"-Attribut wird der String angegeben der mittels des "splitter"-Attributs aufgeteilt wird.

@splitter (optional) (default: ,)

Im "splitter"-Attribut kann der Splitter definiert werden. Aufgrund des Splitters wird der String zerteilt.

@regex (optional)

Wenn eine Regular Expression verwendet werden soll um den String zu zerlegen, kann im "regex"-Attribut eine solche definiert werden.

@param (optional)

Um die aktuelle Position nicht in der Variable $param zu speichern, kann hier ein anderer Variablenname gewählt werden.

Beispiele

```
<art:plain name="data">
   <art:variable name="x" value="Lorem#ipsum#dolor#sit#amet" />
   <art:foreach splitter="#" values="{$x}">
      <art:plain name="word" value="{$param}" />
   </art:foreach>
</art:plain>

Ergebnis (XML):
<data>
   <word value="Lorem" />
   <word value="ipsum" />
   <word value="dolor" />
   <word value="sit" />
   <word value="amet" />
</data>
```

Mit Regular-Expression Splitter:

```
<art:plain name="data">
   <art:variable name="x" value="Lorem#ipsum#dolor#sit#amet" />
   <art:foreach regex="W" values="{$x}">
      <art:plain name="word" value="{$param}" />
   </art:foreach>
</art:plain>

Ergebnis (XML):
<data>
   <word value="Lorem" />
   <word value="ipsum" />
   <word value="dolor" />
   <word value="sit" />
```

```
   <word value="amet" />
</data>
```

art : include

jart.handlers.control.Include

Beschreibung

Die Klasse art:include ermöglicht das Inkludieren anderer JArt-Dateien.

Attribute

Globale Attribute

@href (erforderlich)

Im "href"-Attribut wird der Pfad zur JArt-Datei angegeben die inkludiert werden soll.

Beispiele

```
<art:include href="functions.jart" />
```

art : math

jart.handlers.basic.JArtMath

Beschreibung

Dieser Node bietet verschiedene mathematische Funktionen an, die nicht standardmäßig in XPath integriert sind. Er übergibt der in „name“ festgelegten Variablen den Rückgabewert.

Attribute

Globale Attribute

@name (erforderlich)

Name der Variablen der der Rückgabewert übergeben werden soll

@action (erforderlich)

Folgende Aktionen (Berechnungen) stehen zur Verfügung: Es stehen alle Funtionen der Java Klasse Math zur Verfügung,

```
pi, sin, cos, ceil, floor, abs, atan, atan2, exp, log, pow, round, sqrt, tan, toDegrees,
```

@p1, p2 (optional)

Übergabewerte für die Berechnung

art : plain

jart.handlers.basic.Plain

Beschreibung

art:plain ist eine Klasse mit der man XML-Knoten erzeugen kann. Dieser kann entweder dort erzeugt werden wo der Aufruf geschieht oder in, vor oder nach einem über XPath zu definierenden Knoten. Sollte es notwendig sein, direkt beim Erstellen des XML-Knotens auch Attribute zu erstellen, können diese auch ohne art:attribute angelegt werden, vorausgesetzt der Attributname ist kein reserviertes Attribut.

Attribute

Globale Attribute

@ name *(erforderlich)*

Definiert den Namen des neuen XML-Knotens

@ target *(optional)*

Wird das "target"-Attribut gesetzt, wird der XML-Knoten in dem über XPath definierten XML-Knoten erzeugt.

@ insert-mode *(optional - benötigt target)*

Mit dem "insert-mode"-Attribut kann ein XML-Knoten nicht nur in einem über das "target"-Attribut definierten XML-Knoten erzeugt werden, sondern auch vor oder nach diesem Knoten.

Mögliche Werte:

```
inside, after, before
```

Beispiele

```
<art:plain name="data">
   <art:plain name="node" someattribute="yes"/>
   <art:plain insert-mode="before" target="node" name="newnode" />
</art:plain>

Ergebnis (XML):
<data>
 <newnode />
 <node someattribute="yes" />
</data>
```

art : script

jart.handlers.special.JArtScripting

art:script

Integration vom EcmaScript. Ermöglicht die Erstellung von Logikblöcken mit Zugriff auf alle Java Libraries ohne der Erstellung einer extra Klasse. Die Scripts werden compiliert im Cache gehalten und sind somit auch von der Performance den Java Klassen nicht signifikant Unterlegen. Es kann entweder eine Script Datei (über @script) oder auch ein inline Script (unter Verwendung von @script-id) angegeben werden. Die Handler Klasse steht dabei über $ zur Verfügung.

Steht auch als Extension Function script zur Verfügung

Speziell Funtionsaliase:

- $.E(*name*) >> new org.jdom.Element(*name*)
- $.add(*element*) >> $.outNode.addContent(*element*)

Einbindung von java Packages: importPackage(Packages.*package name*);

- importPackage(Packages.jart);

Attribute

Globale Attribute

@script

Dieses Attribut wird aus Sicherheitsgründen nicht evaluiert und kann daher nicht dynamisch gesetzt werden!

Name der Script Datei. Die Dateiendung bestimmt den Script Typ. (z.b .js für JavaScript).

@script-id

Dieses Attribut wird aus Sicherheitsgründen nicht evaluiert und kann daher nicht dynamisch gesetzt werden!

Innehalb des JART Files und aller dort geladenen Includes Eindeutige ID für das inline Script (wird für die identifizierung des compilierten cache benötigt). Die Endung bestimmt den Script Typ. (z.b .js für JavaScript)

@include

Dieses Attribut wird aus Sicherheitsgründen nicht evaluiert und kann daher nicht dynamisch gesetzt werden!

Beistrichgetrennte Liste der zu Includierenden Scriptdateien,

Beispiele

```
*******************************************************************************
*** JART Code:
*******************************************************************************
<art:script script="sc01.js" for-num="10"/>

*******************************************************************************
```

```
*** Script Code der Datei sc01.js:
********************************************************************************
var forNum = parseInt($.getAtt("for-num"));

for(var i = 0; i < forNum; i++){
    var e = $.E("test-node-x" + i);
    $.add(e);
}

var els = $.filter.selectNodes("/data/*", $.outNode);

for(var i = 0; i < els.size(); i++){
    var e = els.get(i);
    e.setAttribute("checked", "node: " + e.getName());
}


********************************************************************************
*** Ergebniss:
********************************************************************************
<data>
 <test-node-x0 checked="node: test-node-x0"/>
 <test-node-x1 checked="node: test-node-x1"/>
 <test-node-x2 checked="node: test-node-x2"/>
 <test-node-x3 checked="node: test-node-x3"/>
 <test-node-x4 checked="node: test-node-x4"/>
 <test-node-x5 checked="node: test-node-x5"/>
 <test-node-x6 checked="node: test-node-x6"/>
 <test-node-x7 checked="node: test-node-x7"/>
 <test-node-x8 checked="node: test-node-x8"/>
 <test-node-x9 checked="node: test-node-x9"/>
</data>



********************************************************************************
*** Inline Script:
********************************************************************************
<art:script script-id="test01.js">
   var e = $.E("hallo-c");
   $.add(e);
</art:script>
```

art : synced

jart.handlers.control.Synchronized

art:synced

Attribute

Globale Attribute

@ name

Beispiele

art : text

jart.handlers.basic.JArtText

Beschreibung

art:text ist eine Klasse mit der man innerhalb eines XML-Knotens Text erzeugen kann.

Attribute

Globale Attribute

@ value *(erforderlich)*

Im "value"-Attribut wird der auszugebende Text definiert.

@ target

@ append *(optional)*

Wird das "append"-Attribut auf "yes" gesetzt, wird der Text aus dem "value"-Attribut bereits bestehendem Text angefügt.

Beispiele

```
<art:plain name="data">
   <art:text value="Hello" />
   <art:text value=" World!" append="yes" />
</art:plain>

Ergebnis (XML):
<data>Hello World!</data>
```

art : variable

jart.handlers.basic.Variable

art:variable

Beschreibung

Legt Laufzeitvariablen (XPATH Variablen) fest oder modifiziert sie.

Attribute

Globale Attribute

@ name *(erforderlich)*

Name der Variablen

@ value

Wert auf den die Variable gesetzt wird

@ parameter

Parameter 1 für computed-value Operationen

@ parameter2

Parameter 2 für computed-value Operationen

@ property-name

Name der JART Property (jart-config.xml)

@ remove

wenn „yes“ wird die Variable aus den Variablen - Buffer entfernt

@ computed-value

Erstellt einen generierten Wert.

Wert	**Funktion**
long-unique-id	Erstellt eine Globale unique ID
deAccent	
deAccentUrl	
niceImgUrl	
append	hängt die in *value* angegebenen Werte Beistrich-getrennt zusammen. In *parameter* kann ein anderes Trennzeichen angegeben werden.
getNiceUrl	
text	Legt den Text-Knoten unter dem Variablen-Knoten als value ohne XPATH parsing fest.
user-ip	gibt die IP des Clients zurück
unique-id	Erstellt ein pseudo- unique ID. Diese ID ist Systemweit eindeutig und global eindeutig solange nicht ein Server zur selben Millisekunde gestartet wird
substringBeforeLast	letzter String Teil vor dem in parameter angegebenen String
substringAfterLast	letzter String Teil nach dem in parameter angegebenen String
trim	Wendet ein whitespace Parsing auf den Wert an (Java.String.trim())
toUpperCase	wandelt den Wert in Großbuchstaben um
toLowerCase	wandelt den Wert in Kleinbuchstaben um
deEnt	wandelt vorkommende Unicode-Entities Unicode Werte um
regex	wendet die in parameter angegebene regex mit dem im parameter2 angegeben replacement an
regexExt	</nowiki>§)
http-header	gibt den in parameter angegebenen HTTP – Header Teil aus
all-http-header-names	wandelt alle HTTP – Header Teile in Laufzeitvariablen um (name wird ignoriert)
all-request-parameter-names	gibt eine Kommaseparierte Liste der HTTP-Request-Parameter aus
property	Gibt den Wert der JART Property die in property-name angegeben ist aus
decode	Dekodiert den Wert in das in parameter angegebene Format (default: Windows-1252)
convert	Konvertiert ein Bytearray in einen Unicode String

encode	Enkodiert den Wert von dem in parameter angegebene Format (default: Windows-1252) in das Unicode Format
session-id	gibt die aktuelle Session ID aus
server	gibt die Server Domain im URL Format aus
line-feed	gibt ein UNIX Linefeed aus
tab	
now	gibt das aktuelle Datum im Format yyyy-MM-dd HH:mm:ss aus
server-root	
current-folder	gibt die URL zum Übergeordneten Ordner der aktuellen .jart Datei aus
application-root	gibt die URL zum Basisverzeichnis des JART Systems aus
current-path	gibt den JART Pfad URL zum Übergeordneten Ordner der aktuellen .jart Date aus
query-string	gibt den aktuellen HTTP Query String aus
evaluate	wendet ein JART Evaluierung (XPATH) auf den Wert an
dos-line-feed	gibt ein Windows Linefeed (Carrige / Return) aus
random-number	gibt einen Zufallswert zwischen 1 und der in *parameter* angegebenen Zahl zurück
random-tan	gibt einen Zufallsstring mit der in parameter angegebenen länge zurück. In *parameter2* können die Cars angegeben werden aus denen der Zufallsstring bestehen soll (default: 123456789qwertzupaisdfghjkyxcvbnmQWERTZUPASDFGHJKLYXCVBNI
dump-attributes	wandelt alle Attribute des Ausgabeknotens in Laufzeitvariablen mit den Name der Attribute um
attribute-names	gibt eine Kommaseparierte Liste aller Attributname aus
set-rw-hash	
clear-rw-hash	
remove-rw-hash	
get-rw-hash	
encrypt	Wendet eine 3DES Verschlüsselung mit dem in *parameter* angegebenen KEY und dem Server Key an
decrypt	Wendet eine 3DES Entschlüsselung mit dem in *parameter* angegebenen KEY und dem Server Key an
splitPart	gibt den in *parameter* angegebenen Teil des Strings getrennt durch *parameter* zurück
format-number	gibt den Wert formatiert mit dem Zahlenformat in *parameter* (z.B. #,###.00) und der lokalen (default: en) in *parameter2* zurück
escape-sql	
db-escape-sql	
escape-html	
orderUrlParameter	
varsToJSObject	

Beispiele

setzt foo auf den Wert 123

```
<art:valiable name="foo" value="123" />
```

art : wait

jart.handlers.control.Wait

art:wait

Attribute

Globale Attribute

@ wait-for

Beispiele

art : when

jart.handlers.control.When

Beschreibung

Ein art:when Knoten ist mit einer Bedingung versehen und muss sich innerhalb eines art:choose Knotens befinden. Befinden sich mehrere art:when Knoten untereinander wird der erste bei dem die Bedingung wahr ist abgearbeitet.

Attribute

Globale Attribute

@test (erforderlich)

Im Attribut "test" wird die Bedingung angegeben.

Beispiele

```
<art:plain name="data">
   <art:choose>
      <art:when test="1 = 1">
         <art:plain name="node_1" />
      </art:when>
      <art:when test="1 = 1">
         <art:plain name="node_2" />
      </art:when>
      <art:when test="1 = 2">
         <art:plain name="node_3" />
      </art:when>
   </art:choose>
</art:plain>

Ergebnis (XML):
<data>
   <node_1 />
</data>
```

art : while

jart.handlers.control.While

Beschreibung

Die Klasse art:while ist, wie in anderen Programmiersprachen auch, eine Kontrollstruktur um eine Abfolge von Anweisungen auszuführen, bis eine Bedingung erfüllt ist.

Attribute

Globale Attribute

@test (erforderlich)

Im "test"-Attribut wird die Bedingung der While-Schleife angegeben.

Beispiele

```
<art:plain name="data">
   <art:while test="count(node) < 3">
      <art:plain name="node" />
   </art:while>
</art:plain>

Ergebnis (XML):
<data>
   <node />
   <node />
   <node />
</data>
```

art : xml

jart.handlers.basic.Xml

Beschreibung

Die art:xml Klasse ermöglicht das Erzeugen von XML über einen String.

Attribute

Globale Attribute

@ value *(erforderlich)*

Im "value"-Attribut wird der String angegeben aus dem ein XML-Baum erzeugt werden soll.

@ target

@ insert-mode

Beispiele

```
<art:plain name="data">
   <art:xml value="<node><node2>Hello World!</node2></node>" />
</art:plain>

Ergebnis (XML):
<data>
   <node>
      <node2>Hello World!</node2>
   </node>
</data>
```

art:all-http-header

jart.handlers.data.AllHttpHeader

Beschreibung

Die Klasse art:all-http-header liefert verschiedene Information des Clients.

Attribute

Globale Attribute

Beispiele

```
<art:all-http-header />

Ergebnis (XML):
<all-http-header ACCEPT="*/*" ACCEPT-LANGUAGE="de-at" UA-CPU="x86" ACCEPT-ENCODING="gzip,
```

art:cache

jart.handlers.control.Cache

Beschreibung

Löst und setzt JART interne RAM Caches

Attribute

Globale Attribute

@action (optional)

Werte:

Wert	Funktion

release-all	alle RAM Caches werden aufgelöst
release	der Benutzerdefinierten Cache der in name angegeben ist wird aufgelöst
release-all-security	alle Security Caches werden aufgelöst
release-security	der Security Cache der in name angegeben ist wird aufgelöst
release-config	der JART Konfigurationscache wird aufgelöst und die jart-config.xml erneut eingelesen
set	setzt den in select angegebenen Zielknoten in den in name angegebenen Cache
get	liefert den in name angegebenen benutzerdefinierten Cache in das Ausgabedokument an der aktiven Position

@name (optional)

Name des Benutzerdefinierten select Zielknoten für die Operation (XPATH)

art:call

jart.handlers.control.Call

Beschreibung

Mit art:call können Funktionen, welche über art:block definiert wurden, aufgerufen werden. Will man einer Funktion Parameter übergeben, werden diese innerhalb des art:call Knotens angegeben.

Attribute

Globale Attribute

@function (erforderlich)

Hier wird der Funktionsname der Funktion angegeben die aufgerufen werden soll.

Mögliche vordefinierte Werte:

`run-selection`

@select (optional)

@* (optional)

Es können für jeden eigenen Funktionsaufruf Parameter als Attribute mitübergeben werden.

@selection (optional - notwendig, wenn function="run-selection" ist)

Wenn die globale Funktion "run-selection" aufgerufen wird, dann muss in diesem Attribut die db-abb-builder-Selektion (.xml) angegeben werden.

Beispiele

```
<art:call function="dosomething">
   <art:variable name="parameter" value="my value" />
</art:call>
```

```
<art:call function="dosomething" parameter="my value"></art:call>

<art:call function="run-selection" selection="resources/dbcon_appdes/article_list/article
```

art:cookie

jart.handlers.data.JArtCookie

Beschreibung

Liest oder setzt ein Client Cookie

Attribute

Globale Attribute

@name (erforderlich)

Name des Cookies

@action (erforderlich)

Werte:

Wert	Funktion
get	erstellt einen Ausgabe Knoten aus dem über name angegebenen Cookie mit dem Namen cookie und den Attributen name und value
set	erstellt ein Cookie mit dem Wert value und dem Namen name

@value (erforderlich)

Wert der in das Cookie gesetzt werden soll

@max-age (optional)

Maximale Lebensdauer des Cookies in Sekunden

@domain (optional)

Domain für die das Cookie aktiv sein soll

art:db-execute

jart.handlers.data.JArtJdbcExecute

Beschreibung

art:db-execute ermöglicht bei bestehender Datenbankverbindung einen Datenbank Befehl auszuführen.

Attribute

Globale Attribute

@query (erforderlich)

im Attribut "query" wird der Query-String definiert, der ausgeführt werden soll.

Beispiele

```
<art:variable name="new-value" value="15" />
<art:db-execute query="update products set quantity = {$new-value} where products_id = 1
```

art:db-select

jart.handlers.data.JArtJdbcSelect

Beschreibung

Die art:db-select Klasse führt ein Datenbank-Select aus und legt für jede Ergebniszeile einen XML-Knoten an.

Attribute

Globale Attribute

@name (erforderlich)

im Attribut "name" wird der Name der Ergebnis-XML-Knoten definiert.

@query (erforderlich)

im Attribut "query" wird der Query-String definiert, der ausgeführt werden soll.

@start (optional) (default: 0)

Das "start"-Attribut definiert bei welchem Datenbank-Datensatz begonnen werden soll.

@limit (optional) (default: 1000)

Das "limit"-Attribut definiert wie viele Ergebniszeilen maximal selektiert werden sollen.

Beispiele

```
<art:db-select name="result" query="select * from products where product_name LIKE '%bool

Ergebnis (XML):
<result products_id="2" product_name="Magic Book" price="20" />
```

```
<result products_id="7" product_name="Kids Book" price="10" />
<result products_id="12" product_name="book to read" price="5" />


<art:db-select name="result">
select * from products where product_name LIKE '%book%'
</art:db-select>

Ergebnis (XML):
<result products_id="2" product_name="Magic Book" price="20" />
<result products_id="7" product_name="Kids Book" price="10" />
<result products_id="12" product_name="book to read" price="5" />
```

art:debug

jart.handlers.control.DebugMessage

Beschreibung

art:debug ist eine Klasse um Werte für Kontrollzwecke als XML-Kommentar auszugeben.

Attribute

Globale Attribute

@message (erforderlich)

Definiert die auszugebende Nachricht.

Beispiele

```
<art:plain name="data" attr="Hello World!">
   <art:variable name="var" value="Hello Universe!" />
   <art:debug message=" / {$var}" />
</art:plain>

Ergebnis (XML):
<?xml version="1.0" encoding="UTF-8" ?>
<!--
   JArt Messages:
   debug-info: Hello World! / Hello Universe!
-->
<data attr="Hello World!" />
```

art:delete-node

jart.handlers.basic.DeleteNode

Beschreibung

art:delete-node ist eine Klasse mit der man XML-Knoten löschen kann.

Attribute

Globale Attribute

@select (erforderlich)

Mittels des "select"-Attributs kann über XPath der zu löschende XML-Knoten ausgewählt werden.

Beispiele

```
<art:plain name="data">
   <art:plain name="node">
      <art:delete-node select="." />
   </art:plain>
</art:plain>

Ergebnis (XML):
<data />
```

art:element

jart.handlers.basic.JArtElement

Beschreibung

art:element ist wie art:plain eine Klasse mit der XML-Knoten erzeugt werden können. Jedoch können Attribute nicht wie bei art:plain sofort in den art:element Knoten geschrieben werden. Attribute können in einem art:element Knoten nur mittels art:attribute erzeugt werden. Jedoch kann bei art:element - im Gegensatz zu art:plain - ein Namespace prefix vergeben werden.

Attribute

Globale Attribute

@name (erforderlich)

Definiert den Namen des neuen XML-Knotens

@target (optional)

Wird das "target"-Attribut gesetzt, wird der XML-Knoten in dem über XPath definierten XML-Knoten erzeugt.

@insert-mode (optional - benötigt: target)

Mit dem "insert-mode"-Attribut kann ein XML-Knoten nicht nur in einem über das "target"-Attribut definierten XML-Knoten erzeugt werden, sondern auch vor oder nach diesem Knoten.

Mögliche Werte:

```
inside, after, before
```

@prefix (optional)

Das "prefix"-Attribut definiert den Namespace Prefix welcher aber über art:namespace gesetzt werden

muss.

Beispiele

```
<art:element name="data">
   <art:element name="node">
      <art:attribute name="someattribute" value="yes" />
   </art:element>
   <art:element insert-mode="before" target="node" name="newnode" />
</art:plain>

Ergebnis (XML):
<data>
 <newnode />
 <node someattribute="yes" />
</data>
```

art:get-xml

jart.handlers.data.GetXml

Beschreibung

Die Klasse art:get-xml ermöglicht das Laden einer XML-Datei in den aktuellen XML-Baum. Auch das Evalieruen von gesetzten Variablen in der geladenen Datei ist möglich.

Attribute

Globale Attribute

@href (erforderlich)

Das "href"-Attribut gibt den Pfad zu XML-Datei an die geladen werden soll.

@evaluate (optional)

Wird das "evaluate"-Attribut auf "yes" gesetzt, werden Variablen in geschwungenen Klammern in der geladenen Datei geparst.

Beispiele

```
<art:plain name="data">
   <art:variable name="x" value="Hello World!" />
   <art:get-xml evaluate="yes" href="otherfile.xml" />
</art:plain>

Geladenes File (otherfile.xml):
<data>
 <node attr="{$x}" />
</data>

Ergebnis (XML):
<data>
   <data>
      <node attr="Hello World!" />
```

```
   </data>
</data>
```

art:if

jart.handlers.control.If

Beschreibung

art:if ist - wie in jeder anderen Programmiersprache auch - eine bedingte Anweisung.

Attribute

Globale Attribute

@test (erforderlich)

Im "test"-Attribut wird die Bedingung festgelegt.

Beispiele

```
<art:plain name="data">
   <art:if test="1 = 1">
      <art:plain name="node" />
   </art:if>
</art:plain>

Ergebnis (XML):
<data>
   <node />
</data>
```

art:image

jart.handlers.image.JArtImage

Beschreibung

Verändert oder erstellt ein Bild im JPEG, PNG oder GIF Format über JAI

Attribute

Globale Attribute

@name (erforderlich)

Inter Bildbuffer Name mit dem solage nicht die action release aufgerufen wird auf das Bild über art: image zugegriffen werden kann.

@href (optional)

url der Bildqulle (oder Ziel bei action push)

@action (erforderlich)

Werte:

Wert	Funktion
load	Ladet ein in href (URL) definiertes Bild in den Buffer
new	erstellt einen neuen Bildbuffer mit den in width und height angegebenen Dimensionen
width	Breite des Bildbuffers
height	Höhe des Bildbuffers
get-info	gibt die Informationen (Breite, Höhe) des Bildes in einem neuen Ausgabeknoten mit dem Namen image-info aus
push	gibt den Bildbuffer an den Client oder an die in href angegebene URL zurück
release	Löst den Bildbuffer auf und gibt den belegten Speicher frei
rotate-90CCW	dreht den Bildbuffer um 90° gegen den Uhrzeigersinn
rotate-90CW	dreht den Bildbuffer um 90° im Uhrzeigersinn
scale-max	skaliert den Bildbuffer auf maximal die in width und height angegebenen Werten
scale-min	skaliert den Bildbuffer auf minimal die in width und height angegebenen Werten. Ist das Bild größer width und hight wird das Bild verkleinert, ansonsten vergrößert
scale	skaliert den Bildbuffer auf die in width und height angegebenen Werte
put	legt einen zweiten Bildbuffer der über src-img angegeben wird über den in name angegebenen auf der Position x, y mit der in width und heigth definierten Dimension
x	X Koordinate der Operation
y	Y Koordinate der Operation
src-image	zusätzlicher Bildbuffer der Operation

art:jdbc

jart.handlers.data.JArtJdbc

Beschreibung

Erstellt eine JDBC Verbindung und hält sie für alle Code-Unterknoten und Aufrufe aufrecht.

Attribute

Globale Attribute

@user (erforderlich - wenn user, dbcon-config nicht angegeben ist)

@password (erforderlich - wenn user, dbcon-config nicht angegeben ist)

@driver (erforderlich - wenn user, dbcon-config nicht angegeben ist)

@href (erforderlich - wenn user, dbcon-config nicht angegeben ist)

@db-info-fix (optional)

@multiple-connection (optional)

@dbcon-config (erforderlich - wenn user, password, driver, href nicht angegeben ist)

Beispiele

```
<art:jdbc dbcon-config="resources/pfad/zum/dbcon.qcon"></art:jdbc>
```

art:move-node

jart.handlers.basic.MoveNode

Beschreibung

Die Klasse art:move-node ermögicht das Kopieren und Verschieben von XML-Knoten. Beim Kopieren eines XML-Knotens können wahlweise alle Subknoten mitkopiert werden oder es kann nur der Haupt-XML-Knoten kopiert werden.

Attribute

Globale Attribute

@select (erforderlich)

Anhand einer XPath Anweisung wird im "select"-Attribut der Knoten selektiert, der verschoben oder kopiert werden soll.

@target (erforderlich)

Im "target"-Attribut wird mittels XPath festgelegt, in, nach oder vor welchen Knoten der selektierte Knoten verschoben oder kopiert wird.

@copy (optional)

Wird ein "copy"-Attribut angegeben, wird der Knoten nicht verschoben, sondern kopiert. Wenn der Wert "full" angegeben ist, werden alle Subknoten mitkopiert. Beim Wert "simple" wird nur der selektierte Knoten ohne Subknoten kopiert.

Mögliche Werte:

```
full, simple
```

@insert-mode (optional) (default :inside)

Mittels des "insert-mode"-Attributs kann defniert werden, dass der zu verschiebende oder kopierende Knoten nicht in sondern vor oder nach den Ziel-Knoten verschoben oder kopiert wird.

Mögliche Werte:

```
inside, after, before
```

Beispiele

Verschieben

```
<art:plain name="data">
   <art:plain name="node_1" />
   <art:plain name="node_2">
      <art:plain name="node2_1" />
   </art:plain>
   <art:move-node target="node_1" select="node_2" />
</art:plain>

Ergebnis (XML):
<data>
   <node_1>
      <node_2>
         <node2_1 />
      </node_2>
   </node_1>
</data>
```

Kopieren

```
<art:plain name="data">
   <art:plain name="node_1" />
   <art:plain name="node_2">
      <art:plain name="node2_1" />
   </art:plain>
   <art:move-node copy="full" select="node_2" target="node_1" />
</art:plain>

Ergebnis (XML):
<data>
   <node_1>
      <node_2>
         <node2_1 />
      </node_2>
   </node_1>
   <node_2>
      <node2_1 />
   </node_2>
</data>
```

Kopieren

```
<art:plain name="data">
   <art:plain name="node_1" />
   <art:plain name="node_2">
      <art:plain name="node2_1" />
   </art:plain>
   <art:move-node copy="simple" select="node_2" target="node_1" />
</art:plain>

Ergebnis (XML):
<data>
   <node_1>
      <node_2 />
   </node_1>
   <node_2>
      <node2_1 />
   </node_2>
</data>
```

art:namespace

jart.handlers.basic.JArtNamespace

Beschreibung

Legt einen Namespace für das Ausgabe XML an

Attribute

Globale Attribute

@prefix (erforderlich)

Prefix des Namespace

@uri (erforderlich)

URI des Namespace

@target (optional)

Ziel-Knoten im Ausgabedokument

@remove (optional)

wenn mit yes angegeben wird der in uri angegebene Namespace aus dem Dokument oder dem Ziel-Knoten entfernt

art:otherwise

jart.handlers.control.Otherwise

Beschreibung

art:otherwise muss sich ebenfalls in einem art:choose Knoten befinden und wird abgearbeitet wenn keiner der art:when Bedingungen zutrifft.

Beispiele

```
<art:plain name="data">
   <art:choose>
      <art:when test="1 = 2">
         <art:plain name="node_1" />
      </art:when>
      <art:when test="1 = 2">
         <art:plain name="node_2" />
      </art:when>
      <art:otherwise>
         <art:plain name="node_3" />
      </art:otherwise>
   </art:choose>
```

```
</art:plain>

Ergebnis (XML):
<data>
   <node_3 />
</data>
```

art:pdf

jart.handlers.image.JArtPdf2

Beschreibung

Die Klasse art:pdf erzeugt mittels Apache FOP (http://xmlgraphics.apache.org/fop/) PDF-Dateien aus einer XSL-Datei die die FO-Definitonen enthält. Idealerweise wird eine solche XSL-Datei mittels dem in JArt integrierten PDF-Designer erstellt.

Attribute

Globale Attribute

@xsl-file (erforderlich)

Im Attribut "xsl-file" wird die über den PDF-Designer erzeugte XSL-Datei angegeben.

@href (optional)

Mittels des "href"-Attributs wird festgelegt, wohin das erzeugte PDF gespeichert wird. Wird dieses Attribut nicht angegeben, wird das Bild direkt mit dem MIME-Type "application/pdf" an den Browser geschickt.

Beispiele

```
<art:png xsl-file="report.pdfdes.xsl" href="report.pdf" />
```

art:push

jart.handlers.basic.JArtPush

Beschreibung

Die Klasse art:push stellt einen output-stream zum direkten Senden von binären Daten an den Client zur Verfügung.

Attribute

Globale Attribute

@data (optional - benötigt wenn weder init noch finalize gesetzt ist)

Das "data"-Attribut enthält die Daten, die an den Client gesendet werden.

@init (optional)

Das "init"-Attribut initialisiert den Datenstrom und setzt den ContentType standardmäßig auf "text-html".

@finalize (optional)

Mit dem "finalize"-Attribut wird der Datenstrom beendet.

@content-type (optional) (default: text/html)

Mit dem "content-type"-Attribut kann der ContentType des Datenstroms definiert werden. Dieses Attribut funktioniert nur in Verbindung mit dem "init"-Attribut. Eine Übersicht der MIME-Typen ist hier zu finden: http://de.selfhtml.org/diverses/mimetypen.htm

Beispiele

```
<art:push init="yes" />
<art:push data="<html><head></head><body>" />
<art:push data="Hello World!" />
<art:push data="</body></html>" />
<art:push finalize="yes" />
```

art:redirect

jart.handlers.control.Redirect

Beschreibung

Mit art:redirect kann eine serverseitige Weiterleitung durchgeführt werden.

Attribute

Globale Attribute

@href (erforderlich)

Im "href"-Attribut wird die URL definiert auf die weitergeleitet werden soll.

Beispiele

```
<art:redirect href="http://www.jart.at" />
```

art:rename

jart.handlers.basic.Rename

Beschreibung

Mittels art:rename kann man Knoten auf einfache Weise umbenennen. Weiters kann ein Namespace Prefix vergeben werden.

Attribute

Globale Attribute

@ select *(erforderlich)*

Das "select"-Attribut definiert über XPath welcher Knoten umbenannt werden soll.

@ name *(erforderlich)*

Mit dem "name"-Attribut wird der neue Name des selektierten Knotens definiert.

@ prefix *(optional)*

Das "prefix"-Attribut definiert den Namespace Prefix welcher aber über art:namespace gesetzt werden muss.

Beispiele

```
<art:plain name="data">
   <art:plain name="node" />
   <art:rename name="newnode" select="node" />
</art:plain>

Ergebnis (XML):
<data>
   <newnode />
</data>
```

art:save-node

jart.handlers.data.SaveNode

Beschreibung

Die Klasse art:save-node ermöglicht das Speichern eines XML-Knotens in eine Datei oder in eine Variable.

Attribute

Globale Attribute

@select (erforderlich)

Das "select"-Attribut legt fest welcher XML-Knoten selektiert und gespeichert werden soll.

@href (optional - erforderlich wenn to-variable nicht gesetzt)

Mittels des "href"-Attributs legt man den Speicherort und Dateinamen der neuen Datei fest. Der selektierte XML-Knoten wird in diese Datei gespeichert.

@to-variable (optional - erforderlich wenn href nicht gesetzt)

Wenn man das "to-variable"-Attribut angibt, wird der XML-Knoten nicht in eine Datei gespeichert sondern in die Variable, die im "to-variable"-Attribut angegeben ist.

@encoding (optional) (default: Windows-1252)

Mit dem "encoding"-Attribut lässt sich das Encoding der neuen Datei festlegen.

@declarations (optional)

Mittels des "declarations"-Attributs kann man die XML-Declarations per Hand festlegen.

Beispiele

```
<art:plain name="data">
   <art:plain name="node">
      <art:attribute value="test" name="attr" />
   </art:plain>
   <art:save-node encoding="UTF-8" href="newfile.xml" select="." />
</art:plain>

Ergebnis (newfile.xml):
<data>
   <node bla="test" />
</data>
```

art:send-mail

jart.handlers.data.JArtSendMail

Beschreibung

Versendet über den in der JART Config angegebenen Mail Server eine EMail

Attribute

Globale Attribute

@from (erforderlich)

Absender Adresse

@to (erforderlich)

Empfänger Adresse(n) an die das Mail versendet wird. Mehrere werden durch Komma getrennt.

@cc (optional)

CC Adressen an die das Mail versendet wird

@bcc (optional)

BCC Adressen an die das Mail versendet wird

@subject (optional)

Betreff des Mails

@body (optional)

Mail Body (text oder html)

@mime-type (optional)

Mime Type des Mails (default: text/plain; charset="UTF-8")

@attachments (optional)

Komma separierte URL Liste der zu versendenden Dateien

@images (optional)

Komma separierte URL Liste der zu versendenden Bild Dateien die in das Mail eingebettet werden sollen

@auto-add-sources (optional)

wenn „yes“ werden Bild Sourcen automatisch eingebettet

Beispiele

```
<art:send-mail attachments="data/ticketing/{$tic_bestellung_id}/tickets.pdf" body="{$mai
```

art:session

jart.handlers.basic.Session

Beschreibung

Speichert alle Attribute des aktuell Kotens in eine http Session und / oder gibt diese im Aktuellen Knoten aus

art:session-data

jart.handlers.basic.SessionData

Beschreibung

Die Klasse art:session-data erlaubt das Speichern, Holen und Löschen von XML-Knoten in eine dem

Client eindeutig zuweisbare Session.

Attribute

Globale Attribute

@action (erforderlich)

Mittels des "action"-Attributs wird festgelegt, ob eine Session gespeichert, geholt oder gelöscht werden soll. Wir über den Wert "set" eine Session gesetzt muss mit dem Attribut "select" ein XML-Knoten selektiert werden welcher in die Session gespeichert wird.

Mögliche Werte:

```
set, get, clear
```

@name (erforderlich)

Das "name"-Attribut definiert den Session-Namen beim Setzen einer Session und wird auch dazu benützt, die Session beim Holen und Löschen anzusprechen.

@select (optional - benötigt bei Wert "set" von action)

Das "select"-Attribut hat nur Sinn in Verbindung mit dem "set"-Wert des "action"-Attributs. Hier wird mittels XPath festgelegt welcher XML-Knoten in die Session gespeichert wird.

Beispiele

Session speichern

```
<art:plain name="shopping-cart">
 <art:plain name="item" id="3" quantity="1" />
</art:plain>
<art:session-data select="shopping-cart" action="set" name="CART" />
```

Session holen

```
<art:session-data action="get" name="CART" />
```

Session löschen

```
<art:session-data action="clear" name="CART" />
```

art:svg

jart.handlers.image.JArtSVG

Beschreibung

Die Klasse art:svg erzeugt mittels Apache Batik (http://xml.apache.org/batik/) Bilder aus einer XSL-Datei die die SVG-Defintionen enthält. Idealerweise wird eine solche XSL-Datei mittels dem in JArt integrierten SVG-Designer erstellt.

Attribute

Globale Attribute

@image-type (optional) (default: jpg)

Bestimmt in welchem Format das Bild gerendert wird.

Mögliche Werte:

`jpg , png, tiff`

@xsl-file (erforderlich)

Im Attribut "xsl-file" wird die über den SVG-Designer erzeugte XSL-Datei angegeben.

@href (optional)

Mittels des "href"-Attributs wird festgelegt, wohin das erzeugte Bild gespeichert wird. Wird dieses Attribut nicht angegeben, wird das Bild direkt mit dem vom Bildtyp abhängigen MIME-Type (image/jpeg, image/png oder image/tiff) an den Browser geschickt.

@quality (optional) (default: 1)

Kann eine Dezimalzahl zwischen 0 und 1 sein und legt die Qualität eines jpg-Bildes fest. Dieses Attribut hat nur Auswirkung bei jpg-Bildern.

Beispiele

```
<art:svg image-type="png" xsl-file="button.svgdes01.xsl" href="btn.png" />
```

art:sysinfo

jart.handlers.control.JArtCacheInfo

Beschreibung

Gibt einen Ausgabeknoten mit folgenden JAVA Runtime Informationen zurück:

`free-memory, total-memory, max-memory, available-processors`

Attribute

Globale Attribute

@run-gc (optional)

wenn „yes“ wird die JAVA Garbage Collecction aufgerufen

@run-finalization (optional)

wenn „yes“ werden alle Finalization Objekte ausgeführt (kann nur zusammen mit run-gc ausgeführt werden)

art:transform

jart.handlers.basic.XslTransform

Beschreibung

art:transform ist eine Klasse, die es ermöglicht XML mittels einer XSL-Datei zu transformieren. Weiters besteht die Möglichkeit, das durch die Transformation erzeugte XML entweder in eine Datei zu speichern oder direkt in den aktuellen XML-Baum an eine beliebige Stelle zu laden. Bei der Transformation in eine Datei muss nicht zwingend XML erzeugt werden, nahezu jedes beliebige Format ist denkbar.

Attribute

Globale Attribute

@xsl-file (erforderlich)

Im Attribut "xsl-file" muss die XSL-Datei angegeben werden, anhand der XML-Baum transformiert werden soll.

@target (optional)

Das Attribut "target" gibt mittels XPath an, in welchen XML-Knoten das über die Transformation erzeugte XML verschoben werden soll. Wird kein "target" und kein "href"-Attribut verwendet, wird das XML an die Stelle gesetzt, an der der Aufruf geschieht.

@insert-mode (optional - benötigt target) (default: inside)

Mittels des "insert-mode"-Attributs kann in Verbindung mit dem "target"-Attribut ein erzeugtes XML nicht nur in einen XML-Knoten, sondern auch danach oder davor verschoben werden.

Mögliche Werte:

```
after, before, inside
```

@href (optional)

Wird das "href"-Attribut gesetzt, wird das erzeugte XML nicht im aktuellen XML-Baum erzeugt, sondern in die vom "href"-Attribut festgelegte Datei gespeichert.

Beispiele

```
<art:plain name="data">
   <art:plain name="node">
      <art:attribute name="test" value="Hello World!" />
   </art:plain>
   <art:transform href="output.html" xsl-file="main.xsl" />
</art:plain>

XSL-Datei:
```

```
<xsl:template match="/">
   <xsl:for-each select="data">
      <html>
         <head />
         <body>
            <xsl:for-each select="node">
               <span>
                  <xsl:value-of select="@test" />
               </span>
            </xsl:for-each>
         </body>
      </html>
   </xsl:for-each>
</xsl:template>

Ergebnis (HTML-Datei):
<html>
   <head>
      <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
   </head>
   <body>
      <span>Hello World!</span>
   </body>
</html>
```

art:with-node

jart.handlers.control.With

Beschreibung

art:with-node erlaubt das Ansprechen von XML-Knoten mittels XPath von jeder Stelle im Code aus.

Attribute

Globale Attribute

@ select *(erforderlich)*

Das "select"-Attribut legt fest welche XML-Knoten selektiert werden sollen.

Beispiele

```
<art:plain name="data">
   <art:for from="1" to="3" increment="1">
      <art:plain name="node" />
   </art:for>
   <art:with-node select="/*/node">
      <art:attribute value="Hello World!" name="attr" />
   </art:with-node>
</art:plain>

Ergebnis (XML):
<data>
   <node attr="Hello World!" />
   <node attr="Hello World!" />
   <node attr="Hello World!" />
</data>
```

art:zip

jart.handlers.data.JArtZip

Beschreibung

Erstellt oder liest Daten im ZIP Format

Attribute

Globale Attribute

@action (erforderlich)

Wert	Funktion
extract-zip	Extrahiert die in href angegebene ZIP Datei in den in target angegebenen Ordner
create-zip	erstellt eine ZIP - Datei mit der in target angegebenen URL aus den in href angegebenen Dateien. Wird das target attribut ausgelassen wird die Datei direkt an den Client geliefert.
compress	veraltet
extract	veraltet

@href (erforderlich)

URL der Ursprungsdatei(n) oder Ordner. Multiple Werte werden mit Komma separiert.

@target (optional)

URL der Ziel Datei oder Ordner Wenn kein target angegeben ist, dann wird das erstellte zip direkt an den Client gepusht.

@file-name (erforderlich)

Name der Datei die an den Client geliefert wird

Beispiele

```
<art:zip action="create-zip" file-name="name-des-zips" href="resources/.../../order-fuer·
```

JART XSL Extension

Neu

- script

Extension Function: script

- java:script(*script datei, filter, param*)
- java:script(*script id, inline code, filter, param*)

Einbindug von EcmaScript als Extension.

Einbindug wie art : script mit folgenden Ausnahmen:

- **es muss "var result" definiert sein da diese den Rückgabewert festlegt**
- Der $ Scope liegt hier auf dem filter Objekt, da kein Handler zur Verfügung steht.
- Variable root: zeigt auf /* des Ausgabedokuments
- Variable param: übergebener Parameter

Beispiele

```
*****************************************************************************
*** XSL Code
*****************************************************************************
<xsl:stylesheet version="1.0" exclude-result-prefixes="xsl java jart">
 <xsl:output encoding="Windows-1252" method="html"/>
 <xsl:param name="j-j-filter"/>
 <xsl:variable name="script1">
  importPackage(Packages.jart);

  var cob = new CacheObj();
  var result = cob.locked;
 </xsl:variable>
 <xsl:template match="/">
  <html>
   <head>
    <title>Test 4</title>
   </head>
   <body>
    <h1>Test 4</h1>
    <div>
     <xsl:value-of disable-output-escaping="yes" select="java:script("script2.js", $scri
    </div>
    <div>
     <xsl:value-of disable-output-escaping="yes" select="java:script("test4.js", $j-j-fi
    </div>
   </body>
  </html>
 </xsl:template>
</xsl:stylesheet>


*****************************************************************************
*** test4.js:
*****************************************************************************
var result = "test in xsl " + $.params.get("xslFile") + "<br/>";
for(var i = 0; i < parseInt(param); i++){
    result += "[POS:" + i + "]";
}


*****************************************************************************
*** Ergebniss:
*****************************************************************************

Test 4
true
test in xsl test4.xsl
[POS:0][POS:1][POS:2][POS:3][POS:4][POS:5][POS:6][POS:7][POS:8][POS:9][POS:10][POS:11][P
```

v3-zu-v4

Upgrade Projekt zu v4

Im Verzeichnis system im file project.xml das Attribut v4mode="yes" hinzufügen

Standard Startup XML in indexes/$mainindex/packages/standard-startup-v01/standard-startup-v01.xml anpassen:

- xsl:include /prj3/jart-tools/resources/system/blobedit/blob-edit-inc-v3.xsl statt /projects/jart-v3/includes/xsl/blob-edit-inc.xsl
- xsl:param j-j-filter hinzu
- xsl:param self-reference hinzu und im JART Code mit der property self-reference verknüpfen (art:variable computed-value="property" name="self-reference" property-name="self-reference")
- call-template-name="j-edit-start" im template mode="body-startup" einfügen
- call-template-name="j-edit-end" im template mode="body-startup" einfügen
- template blob-info alles im if $blobedit auskommentieren
- call-template name="edit-buttons" auskommentieren
- call-template name="edit-header" auskommentieren
- template match=img-db-img einfügen
- template match=img-db-img-info einfügen
- template match=img-db-img-new einfügen
- template match=img-db-img-src einfügen
- template match=img-db-img-old einfügen
- template match=img-db-img-after einfügen

Falls in standard-layout.xsl custom-xsl-output angegeben darf nicht indent="yes" gesetzt sein. Falls Editierung nicht funktioniert indent auf no setzen.

- project tools (js/css) einmalig öffnen und speichern (wegen Error) und CKEditor Styles einmal öffnen und speichern.

im Standard-startup.jart:

- variable self-reference definieren (computed-value: property; property-name=self-reference)

- doctype checken (!!!!) (bei xalan-projekten)
- Projekt einmal neu auswählen.

Bei alter XML-Bild-DB:

- zuerst in hsqldb konvertieren (+ checken, ob dbcon.xml vorhanden ist)
- im "img-dbcon.xml" Attribut "use-dbtype" (value: /projects/jart-v3/data/dbdef/hsqldb.xml) hinzu (sonst gibts einen Error beim SQL Interface)
- qcon file generieren (fielsystem, senden an)
- Überprüfen ob der Ordner tmp im jart root existiert, wenn ncht den Ordner anlegen

In Bild DB im SQL Fenster unter DB-Execute folgendes Statement ausführen:

```
ALTER TABLE media ADD lookup LONGVARCHAR DEFAULT NULL;
UPDATE media SET lookup = CONCAT(originalname, CONCAT(' ', name));
```

Text-Paket:

- param: allow-in-xdoc (2x) um class und style erweitern (sonst funktionieren die benutzerdefinierten

Stile im Text-Editor nicht)

In den Befüll und Editierbaren Paketen statt blob-edit am Beginn des Html Codes aply-template blobed-start und apply-template blobed-end am ende einfügen. Dann kann das Paket im Editmodus durch klick irgendwo in der Darstellung editiert werden.

Ajax DB Interface auf v4 umstellen (für Texteditor und BildDB) Im dbcon File im ersten Node die folgenden Attribute einfügen:

- v4mode="yes"
- prj="$prj"
- upload-directory="/prj3/prj/data/uploads/"
- link-groups="website:::main:::"