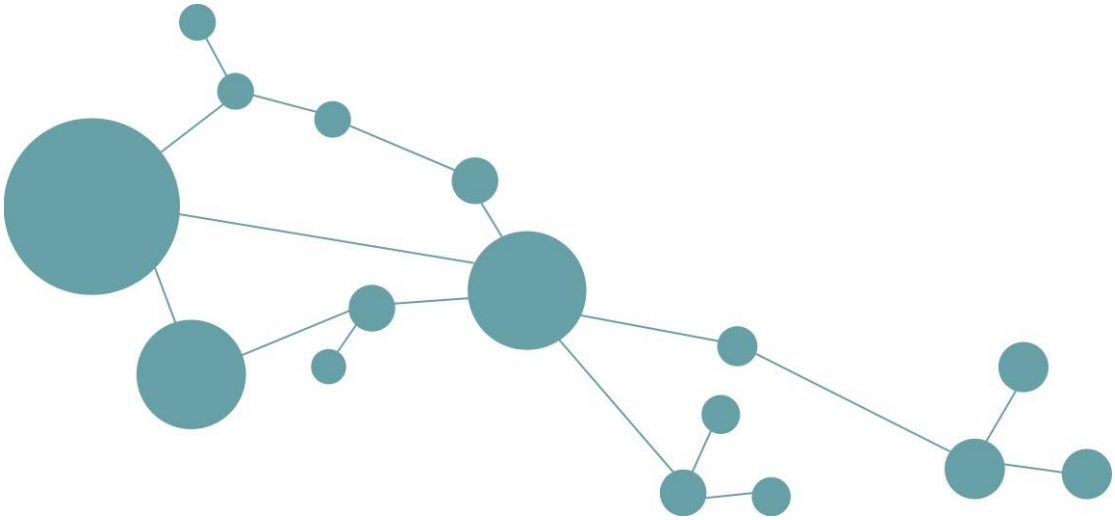


KScript

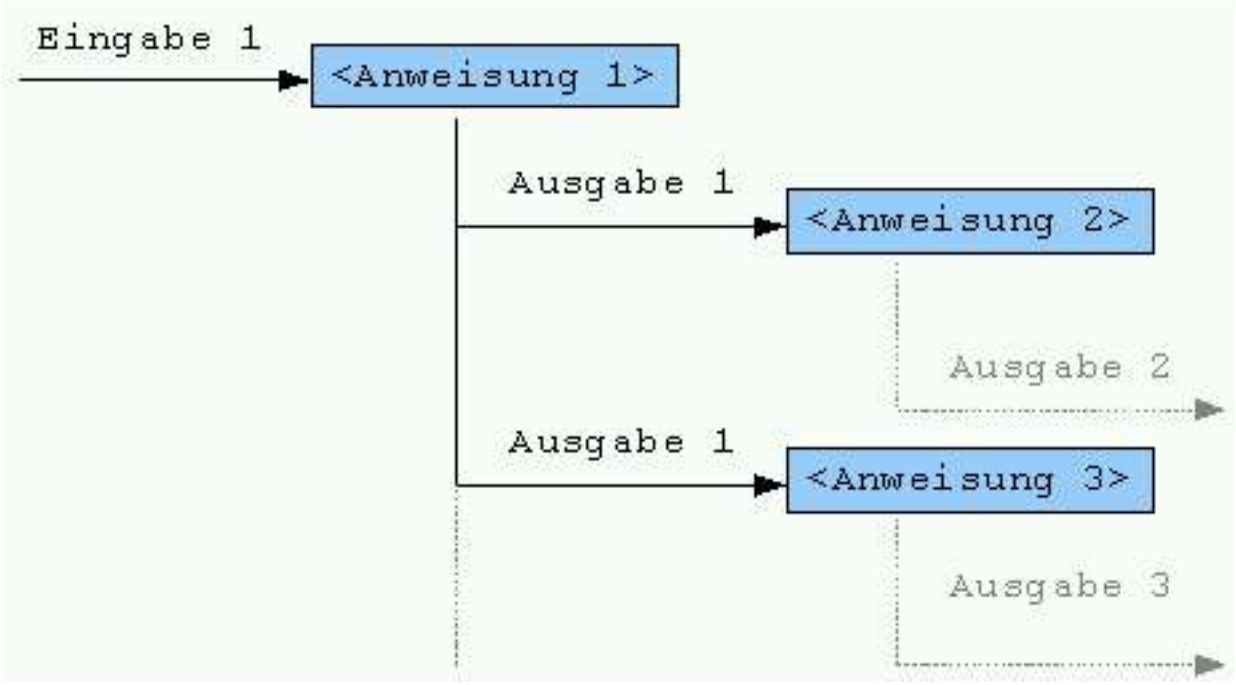
i·views

Inhaltsverzeichnis

1 Überblick über KScript

Mit Skripten kann man sowohl lesend als auch schreibend auf ein Wissensnetz zugreifen. Mit Hilfe von Reportskripten können Daten wie z.B. Attributwerte in eine Textdatei geschrieben werden. Reportskripte werden im Wissensnetz gespeichert und können bei Bedarf allen Benutzern zugänglich gemacht werden. Mit Skripttriggern können bei bestimmten Änderungen am Wissensnetz weitere Aktionen durchgeführt werden (z.B. das Datum der letzten Änderung in einem Attribut speichern).

Skripte werden in XML formuliert und bestehen aus geschachtelten Anweisungen. Jede Anweisung ist im Prinzip ein Baustein, der als Eingabe einen oder mehrere Objekte (Begriff, Attribut, Relation etc.) erhält, diese abarbeitet und Objekte ausgibt, mit denen wiederum untergeordnete Anweisungen ausgeführt werden. Als Eingabe wird immer die Ausgabe des Elternelements verwendet. Alle Nachbarelemente werden mit denselben Eingabenwerten abgearbeitet.



Bei der Abarbeitung wird grundsätzlich zwischen einzelnen Objekten und Mengen von Objekten unterschieden. Manche Anweisungen können nur auf einzelnen Objekte und manche nur auf Mengen angewendet werden. Viele Anweisungen können nur auf bestimmte Typen von Eingabeobjekten (z.B. nur Attribute) angewendet werden. Falls eine Anweisung die Eingabe nicht verarbeiten kann, wird die Abarbeitung mit dem Nachbarelement fortgeführt.

Objekte im Wissensnetz können mit KPath (siehe entsprechendes Kapitel) referenziert werden.

1.1 Beispiele

Wenn das folgende Skript auf ein Ordnerobjekt angewandt wird, gibt es den Inhalt des Ordners aus:

Beispiel Skript 1:

```
<Script>
   <Output>Ordner: </Output><name/><cr/><cr/>
   <FolderElements>
      <Each>
         <Output>Name:</Output><name/><cr/>
      </Each>
   </FolderElements>
</Script>
```

Ausgabe:

```
Ordner: Vitamine

Name: Vitamin B1
Name: Vitamin B2
Name: Vitamin B6
```

Das erste <**Output**> sowie <**Each**> erhalten den Ordner als Eingabe. <**Output**> und <**name**> geben Daten aus. <**Each**> iteriert über eine Menge von Objekten, in diesem Fall alle Elemente des Ordners.

Beispiel Skript 2:

Das folgende Skript angewendet auf den gleichen Ordner gibt nichts aus, da <**Concept**> für Ordner nicht definiert ist und somit die Unterelemente von <**Concept**> nicht mehr ausgeführt werden.

```
<Script>
   <Concept>
      <Output>Begriff: </Output><name/><cr/>
   </Concept>
</Script>
```

2 Konfiguration

Das Root-Element eines Skriptes ist <**Script**>. Über die Attribute dieses Elements kann der Traverser noch konfiguriert werden (alle Attribute sind optional):

Skript	
Parameter:	*Beschreibung:*

[@language]	Kürzel der Standardsprache
[@id]	ID des Objektes, mit dem die Traversierung starten soll
[@encoding]	Encoding für den im Skript ausgegebenen Text (siehe Kapitel Encoding).
[@output]	Ausgabetyp. Momentan unterstützt: • **html**: Erlaubt die Verwendung von HTML-Tags im Skript. HTML-Tags werden in diesem Falle direkt ausgegeben ohne dass ein Output-Kommando notwendig wäre. Hierbei ist zu bedenken, dass die Ausgabe zunächst auf einen internen DOM geschrieben wird. Dieser wird im Hauptspeicher gehalten und erst in eine Datei geschrieben, wenn das Skript abgearbeitet ist. Wenn der DOM zu groß wird, kommt es zu einem "Out-of-Memory"-Error.
[@accessRights]	Boolean: Steuert, ob in der Skriptabarbeitung Zugriffsrechte überprüft werden sollen. Standardwert ist in K-Infinity 2.5 **false**, in K-Infinity 3.1 **true**.
[@transaction]	**single**/**multiple** [default: **single**]: • **single**: Das gesamte Skript wird in einer einzigen Transaktion ausgeführt. • **multiple**: Jede Skriptanweisung wird in gesonderten Transaktionen ausgeführt. Siehe Abschnitt zur Transaktionssteuerung
[@deactivateTriggers]	Boolean: Steuert, ob in der Skriptverarbeitung Trigger aktiviert sind oder nicht. Standardwert: **false**.

Beispiel:

```
<Script language="ger" id="ID1_23456">
   ...
</Script>
```

2.1 Externe Argumente

An das Skript können beim Aufruf auch externe Argumente übergeben werden. Dazu müssen direkt unterhalb von <**Script**> ein oder mehrere Argumente angegeben werden:

```
<argument
   name="name"
   type="string | topic | folder | path"
   default="..."/>
```

- **name** ist der Name des Arguments. Der Wert des Arguments wird unter diesem Namen als globale Variable gespeichert.
- **type** gibt den Typ des Arguments an.
- Im Knowledge-Builder werden bei **topic** und **folder** entsprechende Dialoge zur Auswahl verwendet. Im Script-Tool muss dagegen ein KPath-Ausdruck übergeben werden, der ein passendes Objekt zurückgibt. Bei **path** wird der KPath-Ausdruck ausgewertet und das Ergebnis der Variablen zugewiesen.
- **default** ist ein optionaler Standardwert. Bei den Typen **topic** und **folder** muss ein KPath-Ausdruck angegeben werden, der ein passendes Objekt zurückgibt.

Beispiel:

```
<Script>
   <argument name="name" type="string"/>
   <argument name="person" type="topic"
      default="//Person\Tester/singleElement()"/>
   <argument name="logFolder" type="folder"/>
   <argument name="concepts" type="path"
      default="//Ding/allSubconcepts()"/>
   ...
</Script>
```

2.2 Skript-Bibliotheken

Um Bibliotheken mit KScript-Code anzulegen, kann man Funktionen in registrierten Skripts auslagern. Die Funktionen in registrierten Skripten kann man dann auch aus allen anderen KScripts heraus aufrufen.

Beispiel

Man speichert folgendes Skript in einem registrierten Skript namens "LibSearch":

```
<Script>
   <Function name="eidSearch" arguments="eid">
      <Path
         path="simpleSearch('eidSearch', var(eid))/singleElement()/topic()"/>
   </Function>
</Script>
```

Jetzt kann man in einem anderen Skript (Report, Trigger) dies mit einer der folgenden Methoden aufrufen:

- **Import aller Funktionen des Skriptes**

```
<Script>
   <Import script="LibSearch"/>
   <CreateAttribute
      topic="eidSearch('ID123_456')"
      name="$name$">Foobar</CreateAttribute>
```

```
</Script>
```

- **Expliziter Aufruf von Funktionen in registrierten Skripts**

```
<Script>
   <CreateAttribute
      topic="LibSearch->eidSearch('ID123_456')"
      name="$name$">Foobar</CreateAttribute>
</Script>
```

3 Encoding

Hinweis: Es wird empfohlen das Attribut *encoding* für das Element *Script* nicht zu verwenden. Da beim Schreiben auf einen externen Stream, immer das gleiche, von außen gesetzte, Encoding verwendet werden soll und dieses sich nicht ändert bzw. ändern soll während des Rausschreibens, ist diese Einstellungsmöglichkeit in den meisten Fällen nicht sinnvoll.

Bei Skripts, die Teil eines Reports oder eines Mappings sind, gibt es drei Möglichkeiten, das Encoding einzustellen:

1. Durch die entsprechende Option des Mappings bzw. des Reports (gilt für die gesamte Datei). Nur bei dieser Variante können die eingebauten Viewer die Ausgabe korrekt anzeigen.
2. Im Script-Tag, gilt für das gesamte Skript.
   <**Script encoding="utf-8"**> **....** <**/Script**>
3. Im Output-Tag, gilt nur für den Text innerhalb des Output-Tags.
   <**Output encoding="utf-8"**> **....** <**/Output**>

Encodings, die unterstützt werden:

- iso-8859-*n* (n: zwischen 1 und 15)
- utf-8
- ms-cp-*n* (n: codepage)

Innerhalb eines Skripts kann mit <**encoding/**> das aktuelle Encoding ausgegeben werden.

Mit <**prolog/**> lässt sich für ein Dokument der Prolog ausgeben. Für die Version wird immer 1.0 eingetragen. Für das Encoding wird der Wert des aktuellen Encodings verwendet.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Multi-Byte-Encodings wie UCS-2 und UTF-16 sollte man nicht innerhalb eines Skripts setzen (obwohl es unterstützt wird).

Zur Ausgabe von XML ist es evtl. lohnenswert, im Skript auf **output="html"** umzustellen.

Folgendes Skript gibt XML der Form <**Object**><**Name**>**...**</**Name**></**Object**>

```
<Script output="html" encoding="utf-8">
```

```
   <Object>
      <Name><name/></Name>
      <XMLElement name="Concept">
         <path path="concept()/name()"/>
      </XMLElement>
   </Object>
</Script>
```

Elemente, die nicht zu KScript gehören, werden automatisch als XML-Elemente ausgegeben. Alternativ bzw. bei Namenskonflikten kann man mit <**XMLElement name="tag"**>**...**</**/XMLElement**> ein XML-Element explizit erzeugen.

Attribute kann man mit <**XMLAttribute name="name"**>**...**</**/XMLAttribute**> erzeugen. Achtung: evtl. Probleme bei Mappings, da hier automatisch einzelne Anführungszeichen durch zwei Anführungszeichen ersetzt werden.

```
<XMLElement name="Concept">
   <XMLAttribute name="dmid"><id/></XMLAttribute>
   <path path="concept()/name()"/>
</XMLElement>
```

Mithilfe des Tags <**RawOutput**> lassen sich vorformatierte Inhalte aus Attributen des Wissensnetzes in ein mit html oder xml-Ausgabe gestartetes Script einfügen. Ein Beispiel:

```
<Script output="xml">
    <MyElement>
        <RawOutput><path path="./@preformatted/value()"/></RawOutput>
    </MyElement>
</Script>
```

Der Inhalt des Attributes @preformatted wird direkt ohne Umwandlung innerhalb des Elements "MyElement" übernommen. Auf diese Weise lassen sich z.B. bei der Erzeugung von HTML Textformatierungen direkt wiederspiegeln.

4 Transaktionssteuerung

Ohne weitere Konfiguration wird ein Skript innerhalb einer einzigen Transaktion durchgeführt. Bei umfangreicheren Skripten ist es aus Performancegründen eventuell notwendig, diese auf mehrere Transaktionen zu verteilen.

Dazu muss zuerst für das Skript eingestellt werden, dass mehrere Transaktionen verwendet werden:

```
<Script transaction="multiple">
   ...
</Script>
```

Danach kann man mit Hilfe von <**Transaction**> gezielt Teile des Skriptes in eine Transaktion schachteln:

```
<Transaction>
    <CreateInstance name="Syldavien" concept="//Land">
```

```
        <CreateAttribute name="Einwohner" output="explicit">
            642000
        </CreateAttribute>
    </CreateInstance>
</Transaction>
```

Zusätzlich ist es möglich, mit Hilfe von <**onFailure**> auf einen Fehlschlag der Transaktion (z.B. wegen eines Zugriffkonflikts) zu reagieren:

```
<Transaction>
   <do>
      <CreateInstance name="Syldavien" concept="//Land"/>
   </do>
   <onFailure>
      <output>Rollback<cr/></output>
   </onFailure>
</Transaction>
```

Transaktionen können auch explizit abgebrochen werden. Alle weiteren transaktionsrelevanten Anweisungen innerhalb der Transaktion werden dann ebenfalls nicht mehr ausgeführt:

```
<Script>
   <Transaction>
      <CreateInstance name="Bordurien" concept="//Land"/>
      <AbortTransaction/>
      <CreateInstance name="Syldavien" concept="//Land"/>
   </Transaction>
</Script>
```

In diesem Beispiel wird die Transaktion abgebrochen und keines der beiden Länderobjekte wird angelegt. <**AbortTransaction**> bricht die aktive Transaktion auch ab, wenn die Transaktion nicht durch das Skript gesteuert wird (z.B. in Triggern).

Um beim Iterieren über größere Objektmengen eine vernünftige Transaktionsgröße zu erreichen, kann man bei <**Each**> zusätzlich die Parameter **wrapInTransaction="true"** und **transactionSize="**<***Zahl***>**"** verwenden. Es werden dann jeweils <***Zahl***> Elemente innerhalb einer Transaktion abgearbeitet.

```
<Path path="//Land/instances()">
   <Each wrapInTransaction="true" transactionSize="10">
      <ModifyAttribute attribute="@$lastUpdate$">
         <time/>
      </ModifyAttribute>
   </Each>
</Path>
```

In diesem Beispiel werden alle Individuen des Begriffs "Land" in Gruppen zu je zehn Länderobjekten modifiziert.

5 KPath

Mit KPath können Objekte im Wissensnetz adressiert werden. Die Notation ist an XPath angelegt, unterscheidet sich allerdings in einigen Punkten.

Die einzelnen Elemente des Ausdrucks sind normalerweise mit einem Slash "/" voneinander getrennt. Beginnt der KPath-Ausdruck mit einem "/", startet die Auswertung beim Wurzelbegriff, andernfalls beim aktuellen Objekt (hängt vom Kontext des Aufrufes ab).
Wenn ein Element keinem der aufgeführten Elemente in Tabelle entspricht, wird es als Name eines Unterbegriffs interpretiert. Einfache Namen können ohne Anführungszeichen angegeben werden.

Bei Angabe einer Sprache muss diese über ihr Kürzel gemäß ISO 639-2 ("ger" für Deutsch, "eng" für Englisch, ...) angegeben werden.

Beispiele:

- **@Name**
  Attribut "Name"

- **//Buch\Faust/∼Autor**
  Relation "Autor" des Buches "Faust"

- **//$Artefakt$/Book{eng}**
  Unterbegriff "Book" (englischer Name) des Begriffs "Artefakt" (interner Name)

- **//Buch*[∼Autor/target()/@Name = "Goethe"]**
  Alle Bücher, die von Goethe geschrieben wurden

5.1 Namen

In Verbindung mit **@**, **/**, **//**, \ und \\ können folgende Arten von Namen verwendet werden:

Name	Beschreibung
name	Name in der Standardsprache. Ohne Anführungszeichen muss der Name mit einem Buchstaben, Stern oder Underscore beginnen und darf keine Whitespaces oder für andere Ausdrücke verwendeten Sonderzeichen enthalten. Der Name muss folgenden regulären Ausdruck erfüllen: [a-zA-Z_*][∧/(){}$%{}[],∼@§#+-'"s|∧&]* (Der besseren Lesbarkeit halber wurde das Escape-Zeichen "\" weggelassen)
"*name*" '*name*'	Genügt der Name nicht den obigen Anforderungen, so muss er in einfache oder doppelte Anführungszeichen gesetzt werden. Hierbei dient das Backslash-Zeichen "\" als Escape-Zeichen für etwaig enthaltene Anführungszeichen, z.B. 'Wendy\'s'.
name{lang}	Name in der angegebenen Sprache "lang"

$*name*$, $"*name*"$	Interner Name
§*name*§, §"*name*"§	Systemname
#ID42_1013	ID des Objektes

Namen sind nicht durch Variablen ersetzbar, müssen also direkt im Skript stehen.

5.2 Operatoren

Zahlenwerte können mit den Operatoren **+**, **-**, *****, oder **/** verknüpft werden.

Bei "*", "-" und "/" muss mindestens ein Leerzeichen auf beiden Seiten des Operators stehen.

Klammerung wird unterstützt, z.B. ergibt "(5 + 3) * 4" den Wert 32.

Beispiel: Summe der Relationen von Goethe und Schiller:

```
\\Goethe/~*/size() + \\Schiller/~*/size()
```

Der Operator "+" kann auch zur Verknüpfung von Zeichenketten verwendet werden:

```
//Person\Goethe + " schrieb " + //Buch\Faust
```

ergibt

```
Goethe schrieb Faust
```

Mit dem unären Operator "!" kann ein boolscher Ausdruck negiert werden, z.B.

```
!1=2
```

Für manche Operatoren ist auch eine alternative Schreibweise möglich, die nur aus alphabetischen Zeichen besteht, z.B. "eq" für den Gleichheitsoperator. Bei diesen Schreibweisen muss mindestens ein Leerzeichen zwischen Operator und Operanden stehen. Groß-/Kleinschreibung wird unterschieden, nur klein geschrieben wird der Operator erkannt.

Mögliche Operatoren sind (in absteigender Präzedenz):

Operator	**Alternative Schreibweise**	**Bedeutung**
!	not	Negation (unärer Operator)
*		Multiplikation
/		Division
+		Addition, Verküpfung (nur Zeichenketten)
-		Subtraktion
<	lt	Kleiner als

>	gt	Größer als
<=	le	Kleiner als oder gleich
>=	ge	Größer als oder gleich
=	eq	Gleich
!=	ne	Ungleich
^^	xor	Exklusives Oder (Logischer Operator)
&&	and	Und (Logischer Operator)
||	or	Oder (Logischer Operator)

Da KScript auf XML aufsetzt, müssen für die Angabe der Operatoren '&&', '<' oder '<=' die Zeichen '<' und '&' durch die Entities '<' bzw. '&' ersetzt oder die alternative Schreibweise verwendet werden.

Beispiel "Und":

```
<Path path="var(left) &amp;&amp; var(right)"/>
<Path path="var(left) and var(right)"/>
```

Beispiel "Kleiner als":

```
<Path path="var(left) &lt; var(right)"/>
<Path path="var(left) lt var(right)"/>
```

5.3 Bedingungen

Es können Bedingungen in folgender Form angegeben werden:

path1[*path2*]*path3*

Auf alle Elemente aus ***path1***, die Bedingung ***path2*** erfüllen, wird ***path3*** angewendet.
Um die Bedingung ***path2*** zu formulieren können die Vergleichsoperatoren (siehe voriger Abschnitt) verwendet werden. Boolesche Ausdrücke können mit den boolschen Operatoren verknüpft werden.

Beispiel: Namen aller Bücher, die von Goethe geschrieben wurden

```
//Buch\*[~Autor/target()/@Name = "Goethe"]/@Name/value()
```

6 KScript-Referenz

Dieses Kapitel beinhaltet die Hauptreferenz für KScript und KPath Elemente.

Hinweise:

- Mit "**@name**" bezeichnete Parameter sind Attribute des XML-Elementes bzw. Argumente des KPath-Aufrufs.
- "<**name**>" ist ein Unterelement, "<**name**>*****" mehrere Unterelemente, die beliebig häufig auftreten können. In KPath entfallen diese Elemente bzw. werden auf andere Art und Weise angegeben.
- Optionale Elemente werden durch eckige Klammen umschlossen, z.B. bezeichnet "**[@path]**" ein optionales Attribut mit Namen "path".
- Es wird das XML-Tag für KScript und der KPath-Ausdruck angegeben. Falls einer der beiden Bezeichner nicht angegeben ist, ist die Funktion dort nicht verfügbar.
- Bei KPath können optionale Parameter nur insgesamt weggelassen werden, nicht einzeln. Ausnahme: Falls alle Parameter optional sind, können auch nur die ersten n Parameter angegeben werden, beispielsweise **function(optional1, optional2)** oder **function(optional1)** oder **function()**.
- Anweisungen, die mit "veraltet" gekennzeichnet sind, sollten nicht mehr verwendet werden.
- Alternativnamen werden nur aus Kompatibilitätsgründen unterstützt, sie sollten nicht mehr verwendet werden.
- Bei der Verwendung von KPath-Ausdrücken als Argument von KPath-Aufrufen (bzw. als Attribut einer KScript-Anweisung) wird in einigen Fällen eine Klammerung des KPath-Ausdruckes in **${...}** benötigt. Details hierzu sind im Kapitel zu KPath beschrieben.

Die einzelnen Funktionen werden in den folgenden Kapiteln jeweils in einer Tabelle dargestellt.

Die Kopfzeile jeder Tabelle enthält zwei Textzeilen, wobei die erste Zeile den Namen einer Funktion im KScript-Kontext angibt, die zweite Zeile den Namen in einem KPath-Kontext. Ist eine Funktion in einem Kontext nicht vorhanden, so enthält die entsprechende Zeile ein "-".

Alle Tabellenelemente sind nur vorhanden, wo dies sinnvoll ist.

In einigen Parameterbeschreibungen wird als Beschreibung ein "Namensfilter" genannt, dieses ist ein Suchmuster z.B. der Art "B*".

NameDerFunktionInKScript **nameDerFunktionInKPath(parameter1, [parameter2])**	
Alternativnamen:	alternativerName
Eingabe:	*Effekt/Ausgabe:*
Eingabeobjekt(e)	Ausgabeobjekt(e)/-wert(e)
Parameter:	*Beschreibung:*
@parameter1	benötigter Parameter als Attribut bei KScript verwendet, in der KPath-Funktion durch die Position im oben stehenden Prototyp definiert.
[@parameter2]	optionaler Parameter
<parameter3>	KScript Funktion, die von dieser Funktion eingeschlossen wird.

Optional folgt einer Funktionsbeschreibung noch ein Absatz mit Anwendungsbeispielen und / oder Hinweisen.

Alternativnamen sollten nicht verwendet werden. Statt dessen sollten die im Tabellenkopf angegebenen Funktionsnamen verwendet werden. Diese Dokumentation enthält auch einige mit "(veraltet)" gekennzeichnete Funktionsbeschreibungen. Solche Funktionen bitte nicht mehr verwenden und ggfs. Verwendungen davon durch andere Funktionen ersetzen, da veraltete Funktionen in Zukunft wegfallen können.

Funktionen die "nichts" zurückgeben, geben den undefinierten Wert "null" zurück, der auch zum Vergleichen verwendet werden kann.

Die **Eingabe** ist das Objekt auf dem die Funktion arbeitet. Bei der Eingabe kann es sich sowohl um ein Einzelobjekt als auch um eine Menge von Objekten oder Werten handeln.

Eingabe:	Beschreibung des Eingabeobjektes/Wertes:
-	Das Eingabeobjekt ist egal bzw. wird nicht für die Abarbeitung der Anweisung/Funktion benötigt bzw. verwendet.
Attribut	beliebiges Attribut
Auswahlattribut	Ausprägung eines Attributes vom Typ Auswahl
Begriff	beliebiger Begriff
beliebig	äquivalent zu -
Datumsattribut	Ausprägung eines Attributes vom Typ Datum
Eigenschaft	Attribut oder Relation
Erweiterung	beliebige Erweiterung
Gruppe	äquivalent zu Schlüssel/Wert-Paare
Hit	Ergebnis einer Suche (äquivalent zu Suchtreffer)
Individuum	Ausprägung eines Begriffs oder einer Eigenschaft
Intervall-Attribut	Ausprägung eines Attributes vom Typ Intervall mit beliebigem Attributtyp für die Intervallwerte
Objekt	Alle in einem Wissensnetz vorkommenden Objekte
Objekt (kein Ordner)	Alle Objekte außer den Ordnern
Ordner	beliebiger Ordner
Prototyp	beliebiger Prototyp eines Begriffs, Individuums oder Eigenschaft
Prototypische Eigenschaften	Prototyp vom Typ Attibut oer Relation
Relation	beliebige Relation
Relationsbegriff	Begriff vom Typ Relation

Schlüssel/Wert-Paare	Eine Menge von Paaren (Dictionary), die aus einem Schlüssel und den dazugehörenden Werten bestehen; z.B. durch groupBy() erzeugt
Suchtreffer	äquivalent zu Hit (Ergebnis einer Suche)
Timestamp	Zeitstempel (Datum und Uhrzeit); z.B. durch currentTimestamp() oder timestamp(...) erzeugt
Topic	Begriff, Individuum, Erweiterung, Eigenschaft
Zahl	beliebige Zahl
Zahlenfeld	Eine Menge von Zahlen
Zeichenkette	beliebige Zeichenkette
Zeitstempel-Intervall-Attribut	Ausprägung eines Attributes vom Typ Intervall mit Zeitwerten (Attributtyp: z.B. Datum oder Flexible Zeit)

6.1 Allgemeine Anweisungen

Base64String **base64String([language])**	
Eingabe:	*Effekt/Ausgabe:*
Datei-Attribut	Inhalt des Datei-Attributs als Base64-codierte Zeichenkette.

CurrentLanguage **currentLanguage([format])**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Aktuelle Sprache als iso639-1 Code (z.B. de).
[@format]	Optional kann iso639-2 angegeben werden, um den 3-Buchstaben-Code zu erhalten (z.B. ger).

Id **id()**

Alternativnamen:	IdString, idString()
Eingabe:	*Effekt/Ausgabe:*
Objekt	ID des Objektes als String in der Form IDxxx_yyy. Entspricht der DMID von KEM.

InternalName **internalName()**	
Eingabe:	*Effekt/Ausgabe:*
Objekt	Interner Name des Objekts.

LanguageDo **-**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Setzt die aktive Sprache während der Ausführung der durch <LanguageDo> gekapselten Elemente
Parameter:	*Beschreibung:*
[@language]	KPath-Ausdruck, der die Sprache liefert.

Beispiel: Ausgabe einer Zahl in deutscher Schreibweise

```
<LanguageDo language="ger">
    <path path="@$number$"/>
</LanguageDo>
```

Literal **literal()**	
Eingabe:	*Effekt/Ausgabe:*
Traversierungsergebnis	Das Objekt (Individuum, Begriff, Attribut,Relation), das im Traversierungsergebnis gekapselt ist.

FileSize **fileSize([language])**	
Eingabe:	*Effekt/Ausgabe:*
Datei-Attribut	Größe des Datei-Attributs in Bytes.

GetMimeType **getMimeType()**	
Eingabe:	*Effekt/Ausgabe:*
Datei-Attribut	MimeType der Datei, falls gesetzt.

SetMimeType **setMimeType(mimeType)**	
Eingabe:	*Effekt/Ausgabe:*
Datei-Attribut	Setzt den MimeType.
Parameter:	*Beschreibung:*
@mimeType	MimeType.

Name **name([language])**	
Eingabe:	*Effekt/Ausgabe:*
Objekt	Name des Objekts.
Parameter:	*Beschreibung:*
[@language]	Sprache.

NativeValue nativeValue([language])	
Alternativnamen:	rawValue()
Eingabe:	*Effekt/Ausgabe:*
Attribut	Wert des Attribut. Ausnahme: Bei Blobs der Dateiname.
Anderes Objekt	Das Objekt selbst.
Parameter:	*Beschreibung:*
[@language]	Sprache.

NumericID numericID()	
Eingabe:	*Effekt/Ausgabe:*
Objekt	Numerische ID des Objektes (64-Bit-Ganzzahl).

Path -	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Wertet einen KPath-Ausdruck basierend auf der Eingabe aus.
Parameter:	*Beschreibung:*
@path	KPath-Ausdruck.
[@choose]	**all**/**single**/**any** [default: **all**] : Gibt an, in welcher Form das mengenwertige Resultat des KPath-Ausdrucks zurückgegeben wird: • **all:** alle Resultate • **single:** das einzige Element der Menge oder nichts • **any** ein beliebiges Element oder nichts.

Beispiele:

- Alle Personen:

<**Path path="//Person*"**>

- Alle Personen, die Walter heißen:
  <**Path path="//Person\Walter"**>
- Die Person, die Walter heißt (oder nichts, falls es mehrere oder keine solche Personen gibt):
  <**Path path="//Person\Walter" choose="single"**>
- Alle Unterbegriffe der Eingabe (muss ein Begriff sein)
  <**Path path="./*"**>

Proto **proto()**	
Eingabe:	*Effekt/Ausgabe:*
Topic, Relation, Attribut	Prototyp (Schemadefinition) des Objekts.
Prototyp	Kein Rückgabewert.

Quality **quality()**	
Eingabe:	*Effekt/Ausgabe:*
Hit	Qualität des Hits. Fließkommazahl zwischen 0 und 1.

SortValueString **sortValueString(language)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Für die Elemente der Eingabe werden Anzeigestrings berechnet und diese sind das Ergebnis. Für die Ausgabe wird die Sprache @language, falls angegeben, berücksichtigt
Parameter:	*Beschreibung:*
[@language]	Angabe, in welcher Übersetzung der Ausgabestring gebildet werden soll

SystemName systemName()	
Eingabe:	*Effekt/Ausgabe:*
Objekt	Systemname des Objekts.

User user([name])	
Eingabe:	*Effekt/Ausgabe:*
beliebig	• ohne Name: Benutzer, für den der KPath-Ausdruck ausgewertet wird. • mit Name: Benutzer mit dem angegebenen Namen.
Parameter:	*Beschreibung:*
[@name]	Name des Benutzers.

UserInstance userInstance([name])	
Eingabe:	*Effekt/Ausgabe:*
beliebig	• ohne Name: Benutzerindividuum des Benutzers, für den der KPath-Ausdruck ausgewertet wird. • mit Name: Benutzerindividuum mit dem angegebenen Namen.
Parameter:	*Beschreibung:*
[@name]	Name des Benutzers.

ValueString valueString([language])	
Alternativnamen:	Value, value(), Text, text()
Eingabe:	*Effekt/Ausgabe:*

Attribut	Wert des Attributs, als Zeichenkette.
Anderes Objekt	Name des Objekts.
Parameter:	*Beschreibung:*
[@language]	Sprache.

6.2 Begriffe und Individuen

AllInstances **allInstances([named])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Alle Individuen dieses Begriffes und aller Unterbegriffe. Siehe auch **Instances**.
Parameter:	*Beschreibung:*
[@named]	Namensfilter.

AllSubconcepts **allSubconcepts([named])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Alle Unterbegriffe. Siehe auch **Subconcepts**.
Parameter:	*Beschreibung:*
[@named]	Namensfilter.

AllSuperconcepts **allSuperconcepts([named])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Alle Oberbegriffe. Siehe auch **Superconcepts**.

AllowsExtensions **allowsExtensions()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	• **true**, wenn für das Konzept des Topics (bei Konzepten dieses selbst) Erweiterungsindividuen angelegt werden können. • **false** sonst

AllowsInstances **allowsInstances()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	• **true**, wenn für das Konzept des Topics (bei Konzepten dieses selbst) normale Individuen angelegt werden können. • **false** sonst

Concept **concept()**	
Eingabe:	*Effekt/Ausgabe:*
Individuum, Relation, Attribut	Begriff des Individuums, der Relation oder des Attributs (Attributbegriffe ab K-Infinity 3.0).
Begriff	leeres Ergebnis

ConceptOrSelf **conceptOrSelf()**	
Eingabe:	*Effekt/Ausgabe:*
Individuum, Relation, Attribut	Begriff des Individuums, der Relation oder des Attributs (Attributbegriffe ab K-Infinity 3.0).
Begriff	Der Begriff selbst.

Core **core()**	
Eingabe:	*Effekt/Ausgabe:*
Erweiterung	Gibt das erweiterte Individuum, also den Kern der Erweiterung, zurück.
Begriff, Individuum	Das eingegebenes Objekt selbst, da Begriffe keine Erweiterungen sein können und ein Individuum der Kern selbst ist.

Extensions **extensions([asRelation], [includeConcepts])**	
Eingabe:	*Effekt/Ausgabe:*
Individuum	Gibt die Erweiterungen des Individuums aus.
Begriff	Gibt die erweiternden Begriffe aus (nur bei **includeConcepts**=**true**).
Parameter:	*Beschreibung:*
[@asRelation]	**true**/**false** [default: **false**] • bei **true** werden die Erweiterungen als Relationen vom Individuum zur Erweiterung • bei **false** die Erweiterungen direkt ausgegeben. Für Begriffe gibt es keine solche Relation, es wird eine leere Menge zurückgegeben.
[@includeConcepts]	**true**/**false** [default: **false**]: bei true werden bei Begriffen erweiternde Begriffe zurückgegeben.

InstanceProto **instanceProto()**	
Alternativnamen:	InstProto, instProto()
Eingabe:	*Effekt/Ausgabe:*
Topic, Relation, Attribut	Instanzenprototyp, falls das Objekt bereits ein Prototyp ist, wird es selbst zurück gegeben.

Instances **instances([named])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Alle Individuen des Begriffes aus., bzw. bei erweiternden Begriffen die Erweiterungen. Siehe auch **AllInstances**.
Parameter:	*Beschreibung:*
[@named]	Namensfilter.

- **isKindOf(internalName)**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	• **true**, wenn der Begriff den übergebenen internen Namen hat oder von einem Bergiff mit angegebenen internen Namen erbt. • **false** sonst.
Individuum	Äquivalent zum Ausdruck: **concept()/isKindOf(internalName)**
Parameter:	*Beschreibung:*
@internalName	Interner Name.

Matching **matching([maxTopics], [threshold])** veraltet	
Eingabe:	*Effekt/Ausgabe:*
Topic	Berechnet ähnliche Topics, die in der Umgebung des Topics liegen.
Parameter:	*Beschreibung:*
[@maxTopics]	Maximale Anzahl der Topics. Dies ist aber nur ein Hinweis an den Matching-Algortihmus, die tatsächliche Anzahl der Topics kann auch höher sein.
[@threshold]	Unterer Schwellwert für die Ähnlichkeit (Ganzzahl, 0-100).

NumberOfAllInstances **numberOfAllInstances()**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Anzahl der Individuen dieses Begriffs und aller seiner Unterbegriffe

NumberOfAllSubconcepts **numberOfAllSubconcepts()**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Anzahl aller Unterbegriffe des eingegebenen Begriffs (der Begriff ist nicht mit eingeschlossen!)

NumberOfInstances **numberOfInstances()**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Anzahl der Individuen für genau diesen Begriff (sprich, ohne Unterbegriffe)

Parents **parents()**	
Eingabe:	*Effekt/Ausgabe:*
Topic/Ordner	Gibt für Ordner den Ordner zurück, in dem der aufrufende Ordner liegt. Gibt für Topics das Elternobjekt zurück. D.h. für Begriffe die Menge der Oberbegriffe, für Individuen den Begriff, für Eigenschaften das Topic, an dem sie angebracht sind. Das Ergebnis ist immer eine Menge.

PossibleAttributes **possibleAttributes([named], [includeExtensions])**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Gibt die passenden möglichen Attribute für das eingegebene Topic zurück
Parameter:	*Beschreibung:*
[@named]	Locator, mit dem die Ergebnismenge gefiltert werden kann
[@includeExtensions]	gibt an, ob auch die möglichen Attribute der Erweiterungen miteinbezogen werden sollen, default false

PossibleRelations **possibleRelations([includeExtensions],[defaultRelations],[named])**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Gibt die passenden möglichen Relationen für das eingegebene Topic zurück
Parameter:	*Beschreibung:*
[@includeExtensions]	gibt an, ob auch die möglichen Relationen der Erweiterungen miteinbezogen werden sollen, default false
[@defaultRelations]	Abkürzungsangabe, welche Arten von Relationen zurückgegeben werden sollen. Mögliche Werte: #all #user #restricted #shortcut. default #all
[@named]	Locator, mit dem die Ergebnismenge gefiltert werden kann

PossibleShortcuts **possibleShortcuts()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Sammelt alle möglichen Abkürzungsrelationsprototypen, die von diesem Topic begonnen werden können, auf

Proto **proto()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Prototyp-Objekt für das Topic
Prototyp	der Prototyp selbst

RootConcept **rootConcept()**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt den Wurzelbegriff des Wissensnetzes aus

RootPropertyConcept **rootPropertyConcept()**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt den Wurzelbegriff für Eigenschaften aus

SetAllowsExtendingInstances **setAllowsExtendingInstances(allow, concept)**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Setzt in der Schemadefinition des Begriffs, ob der Begriff erweiterungsfähig ist, auf den Wert @allow. Mit dem KPath-Ausdruck in @concept kann ein anderer zu bearbeitender Begriff gewählt werden
Parameter:	*Beschreibung:*
@allow	KPath-Ausdruck, der den booleschen Wert für das Setzen der Option ergeben muss
@concept	obligater Parameter für die Verwendung in KPath-Ausdrücken zur Bestimmung des Begriffs, der bearbeitet werden soll

SetAllowsInstances **setAllowsInstances(allow, concept)**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Setzt in der Schemadefinition des Begriffs, ob der Begriff individuenfähig ist, auf den Wert @allow. Mit dem KPath-Ausdruck in @concept kann ein anderer zu bearbeitender Begriff gewählt werden
Parameter:	*Beschreibung:*
@allow	KPath-Ausdruck, der den booleschen Wert für das Setzen der Option ergeben muss
@concept	obligater Parameter für die Verwendung in KPath-Ausdrücken zur Bestimmung des Begriffs, der bearbeitet werden soll

Subconcepts **subconcepts([named])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Direkte Unterbegriffe.
Parameter:	*Beschreibung:*
[@named]	Namensfilter.

Superconcepts **superconcepts()**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Direkte Oberbegriffe.

TopAttributeConcept **topAttributeConcept()**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt den Wurzelbegriff für Attribute zurück

TopConcept **topConcept()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Gibt den Wurzelbegriff des Teilnetzes zurück, in dem das Topic liegt.

TopicByID **topicByID(id)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Ermittlung eines Topics anhand der ID
Parameter:	*Beschreibung:*
@id	KPath-Ausdruck zur Ermittlung der ID

TopRelationConcept **topRelationConcept()**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt den Wurzelbegriff für Relationen zurück

CoastIncCounter **coastIncCounter()**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Erhöht den Zähler des Begriffs und gibt den neuen Wert zurück

CoastGetCounter **coastGetCounter()**

Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt den Zähler des Begriffs zurück

6.3 Attribute und Relationen

Attributes **attributes([includeExtensions], [named])**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Attribute des Topics.
Parameter:	*Beschreibung:*
[@includeExtensions]	**true**/**false** [default: **false**]: **true**, falls bei Individuen auch Attribute aller Erweiterungen zurückgeben werden sollen.
[@named]	Namensfilter, nur Attribute diesen Namens werden zurück gegeben.

AttributesInherited **attributesInherited(internalName)** *(ab K-Infinity 3.2.12)*	
Eingabe:	*Effekt/Ausgabe:*
Topic	Liefert eine Menge von Attibuten mit dem angegebenen internen Namen, deren Definition auch vererbt sein kann. Falls keine Attribute des angegebenen Typs am Topic selbst oder an dessen Oberbegriffen definiert sind, wird eine leere Menge zurückgeliefert. Die Funktion unterstützt keine Mehrfachvererbung, d.h. falls an verschiedenen Oberbegriffen unterschiedliche Attribute definiert sind, wird eine leere Menge zurücgeliefert.
Parameter:	*Beschreibung:*
@internalName	String mit dem internen Namen.

AttributeInherited **attributeInherited(internalName)**

Eingabe:	*Effekt/Ausgabe:*
Topic	Liefert ein Attibut mit dem angegebenen internen Namen, dessen Definition auch vererbt sein kann. Falls kein Attribut des angegebenen Typs am Topic selbst oder an dessen Oberbegriffen definiert ist, wird "null" zurückgeliefert. Die Funktion unterstützt keine Mehrfachvererbung, d.h. falls an verschiedenen Oberbegriffen unterschiedliche Attribute definiert sind, wird "null" zurücgeliefert.
Parameter:	*Beschreibung:*
@internalName	String mit dem internen Namen.

AttributeType attributeType()	
Eingabe:	*Effekt/Ausgabe:*

Attribut	Typ des Attributs (als Zeichenkette). Folgende Liste gibt es Ausgabestring je deklariertem Attributstyp an: • Attribut: attribute • Auswahl: choice • Boolesch: boolean • Datei: blob • Datum: date • Datum und Uhrzeit: dateAndTime • Dokument: document • Farbwert: color • Festkommazahl: fixedPoint • Flexible Zeit: flexTime • Fließkommazahl: float • Ganzzahl: integer • Gruppe: container • ID: eid • Interval: interval • Internet-Verknüpfung: url • Passwort: password • Zahl: number • Zeichenkette: string • Zeit: time

ChoicePossibleValues choicePossibleValues([language])	
Eingabe:	*Effekt/Ausgabe:*
Auswahlat-tribut	Liefert die Menge der möglichen Auswahlwerte jeweils mit dem Index als Schlüssel und dem dargestellten Wert in der gewählten Sprache.
Parameter:	*Beschreibung:*
[@language]	optionale Sprache, nur bei übersetzten Choices sinnvoll.

CopyAttribute **copyAttribute(targetPath, [modifyExisting], [language], [withMetas], [property])**	
Eingabe:	*Effekt/Ausgabe:*
Attribut	Kopiert ein konkretes Attribut an das Zielobjekt. Das Anlegen muss natürlich zum Schema des Zielobjekts passen.
Parameter:	*Beschreibung:*
@targetPath	KPath zum Bestimmen des Zielobjekts
[@modifyExist-ing]	**true**/**false** [default: **false**]: bei **false**, werden bereits existierende Attribute dieser Sorte am Zielobjekt nicht verändert.
[@language]	Falls Werte übersetzt sind, gibt **language** die gewünschte Sprache an.
[@withMetas]	**true**/**false** [default: **false**]: bei **true** werden Metaeigenschaften des zu kopierenden Attributs ebenfalls am Attribut des Zielobjekts erzeugt.
[@property]	KPath Asudruck, der das zu kopierende Attribut identifiziert statt des Eingabeattributs.

CopyRelation **copyRelation(targetPath, [modifyExisting], [language], [withMetas], [property])**	
Eingabe:	*Effekt/Ausgabe:*
Relation	Kopiert eine konkrete Relation an das Zielobjekt. Das Anlegen muss natürlich zum Schema des Zielobjekts passen.
Parameter:	*Beschreibung:*
@targetPath	KPath zum Bestimmen des Zielobjekts
[@modifyExist-ing]	**true**/**false** [default: **false**]: Default kann nicht verändert werden.
[@language]	ignoriert
[@withMetas]	**true**/**false** [default: **false**]: bei **true** werden Metaeigenschaften der zu kopierenden relation ebenfalls an der Relation des Zielobjekts erzeugt.
[@property]	KPath Asudruck, der die zu kopierende Relation identifiziert statt der Eingaberelation.

Domains **domains()**

Eingabe:	*Effekt/Ausgabe:*
At- tribut/Relation	Gibt die Menge der Domänen der eingegebenen Eigenschaft zurück

ExtractBlobText **-**	
Eingabe:	*Effekt/Ausgabe:*
Topic/-	Extrahiert aus dem übergebenen oder aufzufindenden Topic am Textattribut @textAttribute für die Sprache @language den Textinhalt der abgelegten Datei
Parameter:	*Beschreibung:*
@textAttribute	Locator zum Auffinden des zu schreibenden Textattributs.
@topic	Topic, an dem der extrahierte Text in das in @textAttribute angegebene Attribut geschrieben wird.
@language	Sprache für übersetzte Textattribute

valueStringGeoFormat(formatKey)	
Eingabe:	*Effekt/Ausgabe:*
GeoAttribut	Gibt die Position in unterschiedlichen Formaten aus
Parameter:	*Beschreibung:*
@formatKey	Eines der vorgegebenen Formate

Vorgegebenen Formate und Ausgabewerte (alle Ausgaben auf der selben Position bis auf CH):

dmn	49° 52.602 N 8° 38.511 E
dmsn	49° 52’ 36" N 8° 38’ 31" E
ndm	N 49° 52.602 E 8° 38.511
ndms	N 49° 52’ 36" E 8° 38’ 31"

KML	KML +8.641855, +49.876704
UTM	32U 474267 5524984
lat	+49.876704, +8.641855
CH	CH 600000 200000 bei Fehlern: CH ### ### und in Klammern der Wert im ndm Format

Inverse **inverse()**	
Eingabe:	*Effekt/Ausgabe:*
Relation	Rückrichtung der Relation.
Relationsbegriff	Rückrichtung des Relationsbegriffs.

IsOneWayInverseRelation **isOneWayInverseRelation()**	
Eingabe:	*Effekt/Ausgabe:*
Relation/Relationsbegriff	Gibt zurück, ob die Eingabe die Rückrichtung einer Einwegrelation repräsentiert

IsSystemComponent **isSystemComponent()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Gibt zurück, ob das Topic ein Systemelement darstellt. Beispiele: password-Attribute, interne Systemrelation wie super/subconcept

IntervalMax **intervalMax()**

Eingabe:	*Effekt/Ausgabe:*
Intervall-Attribut	Gibt die Obergrenze des Intervalls zurück

IntervalMin **intervalMin()**	
Eingabe:	*Effekt/Ausgabe:*
Intervall-Attribut	Gibt die untere Grenze des Intervalls zurück

IntervalSeconds **intervalSeconds()**	
Eingabe:	*Effekt/Ausgabe:*
Zeitstempel-Intervall-Attribut	Gibt die Dauer des Intervalls in Sekunden zurück

IntervalSize **intervalSize()**	
Eingabe:	*Effekt/Ausgabe:*
Intervall-Attribut	Gibt die Differenz aus Obergrenze und Untergrenze des Intervalls zurück

InverseTopic **inverseTopic()**	
Alternativnamen:	target()
Eingabe:	*Effekt/Ausgabe:*

Relation	Relationsziel.
Relationsbegriff	Prototyp des Relationsziels.

InverseTopics **inverseTopics([includeExtensions], [named], [defaultRelations])**	
Alternativename:	RelationTargets
Eingabe:	*Effekt/Ausgabe:*
Relationen	Relationsziele.
Topics	Relationsziele (Relation muss angegeben werden).
Parameter:	*Beschreibung:*
[@includeExtensions]	Siehe **Relations**. Hat keine Bedeutung falls Relationen als Eingabe verwendet werden.
[@named]	Siehe **Relations**. Hat keine Bedeutung falls Relationen als Eingabe verwendet werden.
[@defaultRelations]	Siehe **Relations**. Hat keine Bedeutung falls Relationen als Eingabe verwendet werden.

NameAttributeProto **nameAttributeProto()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Liefert den Inst-Proto desjenigen Attributkonzeptes zurück, welches als Namensattribut für das eingegebene Topic konfiguriert ist. Das Namensattribut eines Topics kann somit über den folgenden KPath-Ausdruck ermittelt werden: "propertiesMatching(nameAttributeProto())"

PossibleRelationTargets **possibleRelationTargets([relation], [targetName])**	
Eingabe:	*Effekt/Ausgabe:*

Topics	mögliche Relationsziele.
Parameter:	*Beschreibung:*
[@relation]	Relationsbegriff oder interner Name des Relationsbegriffes.
[@targetName]	Suchstring für die Filterung möglicher Relationsziele (Eingabe für die Relationszielsuche)

PrimaryTopic primaryTopic()	
Eingabe:	*Effekt/Ausgabe:*
Relation, Attribut	Das Individuum oder der Begriff, an dem die Eigenschaft direkt oder als Metaeigenschaft angebracht ist.
Parameter:	*Beschreibung:*
Topic	Das Topic selbst.

- propertiesMatching(path*1*, ... , path*N*)	
Eingabe:	*Effekt/Ausgabe:*
Topic	Die KPath-Ausdrücke liefern mögliche Eigenschaften des Topics. Es werden dann die dazu passenden tatsächlich vorhandenen Eigenschaften des Topics aufgesammelt und zurückgeliefert.
Parameter:	*Beschreibung:*
@path*X*	KPath, der einen oder mehrere Eigenschaftsprototypen zurückliefert. Als Eingabe wird der Prototyp des Topics verwendet.

Beispiel: alle Relationen vom Typ "kennt" oder "befreundet mit" von Walter

```
//Person\Walter/propertiesMatching(~~kennt, ~~'befreundet mit')
```

PropertyOwner propertyOwner()	
Alternativnamen:	PreferredPropertyOwner, preferredPropertyOwner()

Eingabe:	*Effekt/Ausgabe:*
Eigenschaft	Gibt für die Eingabe das Topic zurück, das als typischer Anbringpunkt der Eigenschaft zu sehen ist. Wichtig für symmetrische Relationen, an denen eine entsprechende Konfiguration angebracht sein sollte, und Einwegrelationen

- possiblePropertiesMatching (path*1*, … , path*N*)	
Eingabe:	*Effekt/Ausgabe:*
Topic	Die KPath-Ausdrücke liefern mögliche Eigenschaften des Topics. Es werden die möglichen Eigenschaften zurückgeliefert, die beim Topic noch zusätzlich angelegt werden können.
Parameter:	*Beschreibung:*
@path*X*	KPath, der einen oder mehrere Eigenschaftsprototypen zurückliefert. Als Eingabe wird der Prototyp des Topics verwendet.

Relations relations([includeExtensions [, defaultRelations [, named]]])	
Eingabe:	*Effekt/Ausgabe:*
Topic	Relationen des Topics.
Parameter:	*Beschreibung:*
[@includeExtensions]	**true**/**false** [default: **false**]: **true**, falls bei Individuen auch Attribute aller Erweiterungen zurückgeben werden sollen.
[@defaultRelations]	Art der zurückzugebenden Relationen, mögliche Werte: all, user, restricted, shortcut, default-Wert: all
[@named]	Namensfilter, nur Relationen diesen Namens werden zurück gegeben.

Topic topic()	
Alternativnamen:	source()

Eingabe:	*Effekt/Ausgabe:*
Relation, Attribut	Das Objekt, an dem die Eigenschaft angebracht ist.
Hit	Topic des Hits.
Topic	Das Topic selbst.
Parameter:	*Beschreibung:*
[@includeExtensions]	**true**/**false** [default: **false**]: **true**, falls bei Individuen auch Relationen aller Erweiterungen zurückgeben werden sollen.
[@named]	Namensfilter für Namen der Relation.
[@defaultRelations]	**all**/**user**/**restricted**. Falls kein Relationsname angegeben wird werden alle Relationen/nur Benutzerrelationen/nur Systemrelationen verfolgt. Falls eine Namensbedingung anegegen wurde, hat dieser Parameter keine Bedeutung.

6.4 Objekte anlegen, modifizieren oder löschen

CreateAttribute **createAttribute(name, [modifyExisting], [language], [output], [skipEmpty], [topic])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff, Individuum	Legt ein neues Attribut an (falls möglich) und setzt dessen Wert. Der Wert des Attributes wird durch die Unterelemente bestimmt (siehe Output).
Parameter:	*Beschreibung:*
@name	KPath-Name des Attributes. Alternativ kann auch mit Hilfe der Syntax "${KPath}" ein KPath-Ausdruck zur Bestimmung des Attributs angegeben werden. Dieser wird ausgehend vom Topic ausgewertet, bei dem das Attribut angelegt werden soll.
[@modifyExisting]	**true**/**false** [default: **false**]: falls kein Attribut existiert, wird ein neues angelegt. • **true**: wenn ein Attribut besteht, wird dieses angepasst. • **false**: wenn ein Attribut existiert, wird dieses nicht angepasst, falls das Attribut mehrfach vorkommen kann, wird ein neues angelegt.
[@language]	Falls Werte übersetzt sind, gibt **language** die gewünschte Sprache an.

[@output]	**always**/**explicit** [default: **always**]: • Bei **always** werden alle Unterelemente von CreateAttribute ausgegeben, also auch Textelemente. • Bei **explicit** muss der Attributwert explizit mit Ausgabeelementen wie z.B. Output ausgegeben werden.
[@skipEmpty]	**true**/**false** [default: **false**]: **true**, falls bei leerem Attributwert kein Attribut angelegt werden soll.
[@topic]	KPath-Ausdruck zur Bestimmung des Topics, das dann statt der Eingabe verwendet wird.

Beispiel: Das Attribut Alter (interner Name "alter") auf 27 setzen oder neu anlegen, wenn es noch nicht vorhanden war.

```
<CreateAttribute
      name="$alter$"
      modifyExisting="true"
      output="explicit">
   <Output>27</Output>
</CreateAttribute>
```

Ab K-Infinity 3.0 gibt es die erweiterte Syntax <**value**> und <**do**>, um das neu erzeugte Attribut weiter verarbeiten zu können:

```
<CreateAttribut name="EID" modifyExisting="true">
   <value>EIDString</value>
   <do>
      <CreateAttribute name="MetaAttribut" output="explicit">
         <Output>MetaAttributString</Output>
      </CreateAttribute>
   </do>
</CreateAttribute>
```

Es wird ein Attribut namens "Metaattribut" mit dem Wert "MetaAttributString" am soeben erzeugten EID-Attribut angelegt.

CreateBlobAttribute **createBlobAttribute(name, [modifyExisting], [language], [topic], filename, [content])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff, Individuum	Legt ein neues Dateiattribut an (falls möglich) und setzt dessen Inhalt.
Parameter:	*Beschreibung:*

@name	KPath-Name des Attributes. Alternativ kann auch mit Hilfe der Syntax "${KPath}" ein KPath-Ausdruck zur Bestimmung des Attributs angegeben werden. Dieser wird ausgehend vom Topic ausgewertet, bei dem das Attribut angelegt werden soll.
[@modifyExisting]	**true**/**false** [default: **false**]: falls kein Attribut existiert, wird ein neues angelegt. • **true**: wenn ein Attribut besteht, wird dieses angepasst. • **false**: wenn ein Attribut existiert, wird dieses nicht angepasst, falls das Attribut mehrfach vorkommen kann, wird ein neues angelegt.
[@language]	Falls Werte übersetzt sind, gibt **language** die gewünschte Sprache an.
[@topic]	KPath-Ausdruck zur Bestimmung des Topics, das dann statt der Eingabe verwendet wird.
@filename	Name des Dateiattributes im Wissensnetz
[@content]	Inhalt, der in das Dateiattribut gespeichert wird. Folgende Typen sind möglich: Typ **ByteArray**: Dieser Typ ensteht z.B. beim Auslesen des Body-Inhaltes eines Rest- Requests, wenn das Objekt kleiner als 8 MBytes ist. Typ **Stream**: Dieser Typ ensteht z.B. beim Auslesen des Body-Inhaltes eines Rest-Requests, wenn das Objekt größer gleich 8 MBytes ist. Typ **Zeichenkette**: Die beliebige Zeichenkette wird als Inhalt des Dateiattributes gespeichert.

Beispiel: Durch einen Request wird eine Datei übermittelt. Mit der folgenden Funktion lässt sich der Inhalt des Request, der im Body enthalten ist, weiterleiten.

```
<Function name="createBlobAttributes" arguments="requestBody,topic">
   <Path path="var(requestBody)">
      <Each>
         <SetVariable variable="content" value="atKey(content)"/>
         <SetVariable variable="filename" value="atKey(filename)"/>
         <If test="var(filename)/size() > 0">
            <do>
               <CreateBlobAttribute
                           name="$datei$"
                           topic="var(topic)"
                           content="${var(content)}"
                           filename="${var(filename)}"/>
            </do>
         </If>
      </Each>
   </Path>
</Function>
```

CreateExtension **createExtension(concept, [core])**	
Eingabe:	*Effekt/Ausgabe:*
Individuum	Es wird eine neue Erweiterung angelegt.
Parameter:	*Beschreibung:*
@concept	KPath-Ausdruck zur Bestimmung des Begriffes der Erweiterung. Der Begriff muss erweiterungsfähig sein.
[@core]	KPath-Ausdruck zur Bestimmung des zu erweiternden Individuums, der dann statt der Eingabe verwendet wird.

CreateInstance **createInstance([name], [language], [concept])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Es wird ein neues Individuum angelegt. Der Begriff muss individuumfähig sein.
Parameter:	*Beschreibung:*
[@name]	Name des Objektes. Falls kein Name angegeben wird erhält der Begriff einen generierten Namen. Alternativ kann auch mit Hilfe der Syntax "${KPath}" ein KPath-Ausdruck zur Bestimmung des Namens angegeben werden.
[@language]	Sprache des Namens.
[@concept]	KPath-Ausdruck zur Bestimmung des Instanzenbegriffes, der dann statt der Eingabe verwendet wird.

CreateRelation **createRelation (name, inverseName, target, [source], [allowMultiple])**	
Eingabe:	*Effekt/Ausgabe:*
Begriff, Individuum	Legt eine neue Relation zu jedem Ziel an (falls möglich), d.h. es wird in jedem Fall auch eine Liste von neu angelegten Relationen ausgegeben.
Parameter:	*Beschreibung:*

@name	KPath-Name der Relation. Alternativ kann auch mit Hilfe der Syntax "${KPath}" ein KPath-Ausdruck zur Bestimmung der Relation angegeben werden. Dieser wird ausgehend von der Relationsquelle ausgewertet.
@inverseName	Name der inversen Relation. Optional, wird nicht benötigt wenn die inverse Relation eindeutig durch das Schema bestimmt werden kann. Alternativ kann auch mit Hilfe der Syntax "${KPath}" ein KPath-Ausdruck zur Bestimmung der Relation angegeben werden. Dieser wird ausgehend vom Relationsziel ausgewertet.
@target	KPath-Ausdruck zur Bestimmung des Ziels. Target wird bezüglich der Quelle der Relation berechnet, wenn also Source angegeben ist, nicht unbedingt bezüglich der Eingabe von CreateRelation.
[@source]	KPath-Ausdruck zur Bestimmung der Quelle, die dann statt der Eingabe verwendet wird.
[@allowMultiple]	**true**/**false** [default: **false**]: **true**, falls eine weitere Relation angelegt werden soll, wenn bereits eine passende Relation von Quelle nach Ziel vorhanden ist.

Beispiel: Walter lernt Goethe kennen.

```
<CreateRelation
    source="//Person\Walter"
    name="kennt"
    target="//Person\Goethe"/>
```

Hinweis: In älteren Version müssen, um die neu erzeugte Relation weiterzuverarbeiten, die Unterelement durch <**do**> eingeschlossen werden:

```
<CreateRelation
      source="//Person\Walter"
      name="kennt"
      target="//Person\Goethe"/>
   <do>
      ...
   </do>
</CreateRelation>
```

CreateSubconcept **createSubconcept([name], [language], [superconcept])**	
Eingabe:	*Effekt/Ausgabe:*

Begriff	Es wird ein neuer Unterbegriff angelegt. Die Funktion ist nur für normale Begriffe verwendbar. Unterbegriffe von Eigenschaftsbegriffen können nicht erzeugt werden.
Parameter:	*Beschreibung:*
[@name]	Wie bei **CreateInstance**
[@language]	Wie bei **CreateInstance**
[@supercon-cept]	KPath-Ausdruck zur Bestimmung des Oberbegriffes, der dann statt der Eingabe verwendet wird.

Beispiel:

```
<CreateSubconcept
    name="painter"
    language="eng"
    superconcept="//Künstler"/>
```

Delete **delete([path])**	
Eingabe:	*Effekt/Ausgabe:*
Objekt (kein Ordner)	Löscht das Objekt.
Parameter:	*Beschreibung:*
[@path]	KPath-Ausdruck zur Bestimmung des Objekts, das dann statt der Eingabe verwendet wird.

DeleteTranslation **deleteTranslation([language], [attribute])**	
Eingabe:	*Effekt/Ausgabe:*
Attribut	Entfernt die in **language** angegebene Übersetzung, falls vorhanden. In K-Infinity 2 wird beim Entfernen der letzten Übersetzung am Ende der Transaktion auch das Attribut gelöscht.
Parameter:	*Beschreibung:*
[@language]	Die Sprache, die gelöscht werden soll. Ohne diese Angabe wird die aktuelle Sprache verwendet.

[@attribute]	KPath-Ausdruck zur Bestimmung des Attributes, das dann statt der Eingabe verwendet wird. Falls der Ausdruck mehr als ein Attribut oder gar keins liefert, wird keine Änderung vorgenommen.

ModifyAttribute **modifyAttribute([language], [skipEmpty], [output], [attribute])**	
Eingabe:	*Effekt/Ausgabe:*
Attribut	Modifiziert den Wert eines Attributes.
Parameter:	*Beschreibung:*
[@language]	Falls Werte übersetzt sind, gibt **language** die gewünschte Sprache an. Ohne diese Angabe wird die aktuelle Sprache verwendet.
[@skipEmpty]	**true**/**false** [default: **true**]: Falls **true**, wird das Attribut nicht modifiziert, wenn der Attributwert leer ist.
[@output]	**always**/**explicit** [default: **always**]: • Bei **always** werden alle Unterelemente von ModifyAttribute ausgegeben, also auch Textelemente. • Bei **explicit** muss der Attributwert explizit mit Ausgabeelementen wie z.B. Output ausgegeben werden.
[@attribute]	KPath-Ausdruck zur Bestimmung des Attributes, das dann statt der Eingabe verwendet wird. Falls der Ausdruck mehr als ein Attribut oder gar keins liefert, wird keine Änderung vorgenommen.

Beispiel:

```
<ModifyAttribute attribute="@Kommentar" output="explicit">
    Letzte Änderung am <date/>
</ModifyAttribute>
```

Auch hier lässt sich durch Angabe von <**value**> und <**do**> Unterelementen das zu verändernde Attribut weiter bearbeiten:

```
<ModifyAttribute attribute="@Kommentar" output="explicit">
   <value>Letzte Änderung am <date/></value>
   <do>
      ...
   </do>
</ModifyAttribute>
```

Mit Hilfe von KPath-Operatoren können auch Berechnungen durchgeführt werden. Die folgende Anweisung erhöht einen Zähler um 1:

```
<ModifyAttribute attribute="@$counter$" output="explicit">
   <path path="@$counter$/rawValue() + 1"/>
</ModifyAttribute>
```

SetSuperConcepts setSuperConcepts (superconcepts)	
Eingabe:	*Effekt/Ausgabe:*
Begriff	Ersetzt die Oberbegriffe von <Concept> durch die im Parameter angegebenen Begri
Parameter:	*Beschreibung:*
@superconcepts	KPath-Ausdruck, der die Menge der neuen Oberbegriffe angibt

RelocateRelation relocateRelation (target)	
Eingabe:	*Effekt/Ausgabe:*
Relation	Legt einen neue Relation zu dem spezifizierten Ziel als Kopie der gegebenen Rela
Parameter:	*Beschreibung:*
@target	KPath-Ausdruck zur Bestimmung des Ziels.

6.5 Zugriffsrechte prüfen

Allows **allows(operation)**	
Alternativnamen:	CanDo
Eingabe:	*Effekt/Ausgabe:*
Topic	Führt die Rechteprüfung auf Operation @operation auf dem Topic aus und gibt true/false aus. User muss gesetzt sein!
Parameter:	*Beschreibung:*

@operation	benötigter Parameter als Attribut bei KScript verwendet, in der KPath-Funktion durch die Position im oben stehenden Prototyp definiert.

- canCreatePropertyAt(topicPath, [language])	
Eingabe:	*Effekt/Ausgabe:*
Prototypische Eigenschaften	Filtered alle Eigenschaften heraus, die an dem Topic wegen mangelnder Rechte nicht angelegt werden könnten.
Parameter:	*Beschreibung:*
@topicPath	KPath Ausdruck, der das Zieltopic berechnet.
[@language]	optionale Sprache für die die Überprüfung stattfinden soll.

CanDo canDo(operation)	
Alternativnamen:	Allows, allows
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Liefert **true**, falls der angemeldete Benutzer die Operation auf alle Objekte der Eingabemenge ausführen darf, anonsten **false**.
Parameter:	*Beschreibung:*
@operation	Name der Operation, die für die Eingabemenge überprüft werden soll.
[@userPath]	optionaler KPath Ausdruck, der das User-topic berechnet, für das die Abfrage durchgeführt werden soll.

- canModifyPropertyAt(topicPath, [language])	
Eingabe:	*Effekt/Ausgabe:*
Eigenschaften	Filtert alle Eigenschaften heraus, die an dem Topic wegen mangelnder Rechte nicht modifiziert werden könnten.
Parameter:	*Beschreibung:*

@topicPath	KPath Ausdruck, der das Zieltopic berechnet.
[@language]	optionale Sprache für die die Überprüfung stattfinden soll.

SelectCanDo **selectCanDo(operation)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Selektiert die Objekte aus der Menge, für die der aktuelle Nutzer das Recht hat, die Operation @operation auszuführen
Parameter:	*Beschreibung:*
@operation	Symbol zur Angabe der Operation
[@userPath]	optionaler KPath Ausdruck, der das User-topic berechnet, für das die Abfrage durchgeführt werden soll.

6.6 Ordner

AddToFolder **addToFolder([folder])**	
Eingabe:	*Effekt/Ausgabe:*
Topics	Legt ein oder mehrere Topics in einem Ordner ab.
Parameter:	*Beschreibung:*
[@folder]	KPath-Ausdruck, der den Ordner (statt des Standardordners) bestimmt. Ohne diese Angabe wird der Standardordner verwendet.

ClearFolder **clearFolder([folder])**	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Löscht den Ordnerinhalt.
Parameter:	*Beschreibung:*
[@folder]	KPath-Ausdruck, der den Ordner (statt des Standardordners) bestimmt. Ohne diese Angabe wird der Standardordner verwendet.

CreateFolder **createFolder(name, [parent])**	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Es wird ein neuer Unterordner angelegt.
Andere Objekte	Es wird ein neuer Ordner angelegt. Der übergeordnete Ordner kann durch den Parameter **parent** angegeben werden.
Parameter:	*Beschreibung:*
@name	Name des neuen Ordners. Alternativ kann auch mit Hilfe der Syntax "${KPath}" ein KPath-Ausdruck zur Bestimmung des Namens angegeben werden.
[@parent]	KPath-Ausdruck zur Bestimmung des übergeordneten Ordners. Falls dieser Ausdruck nicht angegeben ist und die Eingabe kein Ordner ist, wird der Standardordner verwendet. Einen schnelleren Zugriff auf den übergeordneten Ordner erreicht man mit derAngabe einer externen ID der Form id. Dadurch entfällt auch die Angabe des Pfades des übergeordneten Ordners.

Beispiele:

```
<CreateFolder
   parent="folder(Arbeitsordner)"
   name="Unterordner"/>

 <CreateFolder
   parent="folder($externeIDDesOrdners$)"
   name="Unterordner"/>

 <CreateFolder
   parent="folder(Arbeitsordner)"
   name="${concat('_','Unterordner',var(parentname))}"/>
```

FindOrCreateFolder **findOrCreateFolder(name, [parent])**	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Es wird ein Unterordner mit dem angegebenen Namen zurückgegeben. Falls ein solcher Unterordner noch nicht existiert wird ein entsprechender Unterordner angelegt.
Andere Objekte	Es wird ein Ordner gefunden oder nötigenfalls angelegt. Der übergeordnete Ordner kann durch den Parameter **parent** angegeben werden.
Parameter:	*Beschreibung:*
@name	Name des gewünschten Ordners (KPath oder String).
[@parent]	KPath-Ausdruck zur Bestimmung des übergeordneten Ordners. Falls dieser Ausdruck nicht angegeben ist und die Eingabe kein Ordner ist, wird der Standardordner verwendet. Einen schnelleren Zugriff auf den übergeordneten Ordner erreicht man mit derAngabe einer externen ID der Form id. Dadurch entfällt auch die Angabe des Pfades des übergeordneten Ordners.

Folder **folder(name, [isExternalID])**	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Direkter Unterordner mit dem angebeben Namen.
beliebig	Alle Ordner mit dem angebeben Namen, die innerhalb des Arbeitsordners liegen. Es werden sämtliche Unterordner durchsucht.
Parameter:	*Beschreibung:*
@name	Name des Ordners. Bei Angabe einer Zeichenkette werden die Namen der Ordner durchsucht. Bei Angabe einer externen ID der Form id wird der Ordner mit der passenden ID zurückgeliefert. Alternativ kann mit ${KPath} auch ein KPath-Ausdruck zur Bestimmung des Namens angegeben werden. Vom Parameter isExternalID hängt dann ab, wie der Name ausgewertet wird.
[@isExternalID]	Optionaler Boolescher Wert, der bei einem KPath-Ausdruck angibt, ob der Name eine externe ID (true) oder ein Ordnername (false) ist. Der Standardwert ist true.

Beispiel:

```
<Folder name="Ordner">
    <Each>
        <Output><name/><cr/></Output>
    </Each>
</Folder>

<path path="folder($ordnerId$)/label()"/>
```

FolderElements **folderElements()**	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Alle Topics, die sich in dem Ordner befinden. Falls der Ordner ein Expertensuchordner ist, wird die Suche ausgeführt und das Ergebnis geliefert.

GetExternalID **getExternalID()**	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Liefert die externe ID des Ordners

Parents **parents()**	
Eingabe:	*Effekt/Ausgabe:*
Topic/Ordner	Gibt für Ordner den Ordner zurück, in dem der aufrufende Ordner liegt. Gibt für Topics das Elternobjekt zurück. D.h. für Begriffe die Menge der Oberbegriffe, für Individuen den Begriff, für Eigenschaften das Topic, an dem sie angebracht sind. Das Ergebnis ist immer eine Menge.

PublicFolder **publicFolder()**

Eingabe:	*Effekt/Ausgabe:*
beliebig	Liefert den öffentlichen Arbeitsordner (unabhängig von der konkreten Bennenung dieses Ordners).

PrivateFolder privateFolder()	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Liefert den privaten Ordner des Benutzers.

RemoveFromFolder removeFromFolder([folder])	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Entfernt Topics aus einem Ordner.
Parameter:	*Beschreibung:*
@folder	KPath-Ausdruck, der den Ordner (statt des Standardordners) bestimmt. Ohne diese Angabe wird der Standardordner verwendet.

SetExternalID setExternalID()	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Setzt die externe ID des Ordners
Parameter:	*Beschreibung:*
@id	KPath-Ausdruck zur Bestimmung der ID

Beispiel:

```
<Path path="publicFolder()/folder(Testordner)">
    <SetExternalID id=" 'external.' + label()"/>
</Path>
```

SubFolders **subFolders()**	
Eingabe:	*Effekt/Ausgabe:*
Ordner	Liefert die Unterordner des/der Ordner

6.7 Variablen

CopyVariable **copyVariable(source, target)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Weist der Variablen **target** den Wert der Variablen **source** zu. Die Eingabe wird zurückgegeben.
Parameter:	*Beschreibung:*
@source	Name der Quellvariable.
@target	Name der Zielvariable.

DeclareVariable **-**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Deklariert eine lokale Variable. Der Wert der Variable kann analog zu **SetVariable** belegt werden. Hinweise: • Lokale Variablen des Hauptblocks eines Skriptes sind in Funktionen nicht sichtbar. • Wenn gleichnamige globale Variablen existieren, haben lokale Variablen Vorrang, d.h. sie überschreiben lokal die globale Variable. • Each-Blöcke stellen keinen eigenen Kontext für Variablen zur Verfügung, daher verursachen DeclareVariable-Aufrufe innerhalb von Each einen Fehler ab dem zweiten Betreten des Each-Blocks. Daher sollten Variablen außerhalb von Each deklariert werden.

Parameter:	*Beschreibung:*
@variable	Name der Variable.
[@value]	KPath-Ausdruck, der den Wert der Variable bestimmt.

SetVariable **setVariable(variable, value)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Setzt den Wert einer Variablen. Der Wert ergibt sich in KScript aus dem Ergebnis der untergeordneten Anweisungen. Falls eine oder mehrere der untergeordneten Anweisungen Werte ausgeben, wird die gesamte Ausgabe als Wert verwendet. Falls keine Ausgabeanweisungen vorhanden sind, wird der Rückgabewert der letzten Anweisung verwendet. In KPath muss ein zweiter Path zur Bestimmung des Wertes angegeben werden. Falls die Variable nicht deklariert wurde, wird sie implizit als globale Variable gesetzt. Dies ist besonders bei Variablen innerhalb von Funktionen zu beachten, üblicherweise sollte man Variablen in Funktionen immer deklarieren vor dem Benutzen. Es wird der Eingabewert zurückgegeben.
Parameter:	*Beschreibung:*
@variable	Name der Variable.
[@value]	KPath-Ausdruck, der den Wert der Variablen bestimmt. Bei KScript optional, bei KPath obligatorisch. Bei KScript werden bei Angabe dieses Attributs die untergeordnete Anweisungen ignoriert.

Beispiel: Neuanlage einer Relation zwischen einem Objekt und einem neu erzeugten zweiten Objekt:

```
<Script>
   <SetVariable variable="source" value="."/>
   <CreateInstance concept="//$Person$" name="Otto">
      <CreateRelation name="$knows$" target="var(source)"/>
   </CreateInstance>
</Script>
```

SetVariableValue -	
Alternativer Name:	SetVariableObjects
Eingabe:	*Effekt/Ausgabe:*
beliebig	Setzt den Wert einer Variable. Als Wert wird der Rückgabewert der untergeordneten Anweisungen verwendet. Es wird der Eingabewert zurückgegeben.
Parameter:	*Beschreibung:*
@variable	Name der Variable.

Beispiel:

```
<Script>
   <SetVariableValue variable="source">
      <Path path="."/>
   </SetVariableValue>
   <CreateInstance concept="//$Person$" name="Otto">
      <CreateRelation name="$knows$" target="var(source)"/>
   </CreateInstance>
</Script>
```

SetVariableFromOutput -	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Setzt den Wert einer Variable. Als Wert wird der Rückgabewert der untergeordneten Anweisungen verwendet. Es wird der Eingabewert zurückgegeben.
Parameter:	*Beschreibung:*
@variable	Name der Variable.
@value	KPath-Ausdruck, der zur Bestimmung des Wertes ausgewertet wird
[@output]	**implicit, explicit** [default: **implicit**]: • Bei **implicit** werden alle Unterelemente ausgegeben, also auch Textelemente. • Bei **explicit** muss der Wert explizit mit Ausgabeelementen wie z.B. Output ausgegeben werden.

Unterschiede

Mit den vier eben genannten Ausdrücken lassen sich Variablen setzen. In der folgenden Tabelle sind die Unterschiede tabellarisch zusammengestellt.

-: Die Funktion wird nicht angeboten bzw. sollte nicht verwendet werden.

o: Diese Funktion steht zur Verfügung, sollte aber besser durch einen anderen Aufruf durchgeführt werden.

Ausdruck	**Variablentyp**	**Ermittlung des Wertes** (value)		
		KScript-Ausdruck mit ...		**KPath-Ausdruck**
		Attributen	**Unterelementen**	
DeclareVariable	lokal	zweiter Path-Ausdruck	Ergebnis der untergeordneten Anweisungen	-
SetVariable	global	zweiter Path-Ausdruck	o	zweiter Path-Ausdruck
SetVariableValue	global	o	Ergebnis der untergeordneten Anweisungen	-
SetVariableFromOutput	global	-	Untergeordnete Anweisungen liefern Inhalte. Berücksichtigung einer impliziten und expliziten Ausgabe	-

Beispielskript

```
<Script>
    <SetVariable variable='varName1' value='varWert1'/>
    <Output><path path="var(varName1)"/></Output>
    <cr/>
    <Path path="var(varName1)/setVariable(varName2,.)"/>
    <Output><path path="var(varName2)"/></Output>
    <cr/>
    <SetVariableValue variable="varName3">
        <Path path="'varWert3'"/>
    </SetVariableValue>
    <Output><path path="var(varName3)"/></Output>
    <cr/>
    <SetVariableFromOutput variable="varName5">
        varWert5 - <Output>varWert5</Output>
    </SetVariableFromOutput>
    <Output><path path="var(varName5)"/></Output>
```

```
    <cr/>
    <SetVariableFromOutput variable="varName6" output="explicit">
        varWert6 - <path path="varWert6"/>
    </SetVariableFromOutput>
    <Output><path path="var(varName6)"/></Output>
</Script>
```

Ausgabe des Beispielskriptes

```
varWert1
varWert1
varWert3
varWert5 - varWert5
varWert6
```

UserVariable userVariable(named)	
Alternativnamen:	userVar(named)
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt den Wert der Variablen @named zurück, welche zuerst auf dem Transaktionskontext und danach auf dem Benutzerkontext gesucht wird
Parameter:	*Beschreibung:*
@named	KPath-Ausdruck zur Bestimmung des Variablennamens

Variable var(named)	
Alternativnamen:	variable(named)
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt den Wert der Variable zurück.
Parameter:	*Beschreibung:*
@named	Name der Variable. Mit Hilfe der Syntax "${KPath}" kann ein KPath-Ausdruck zur Bestimmung des Variablennamens angegeben werden.

6.8 Funktionen

Call **call(function, parameter*1*, ..., parameter*N*)** <***functionName***> **(parameter*1*, ..., parameter*N*)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Aufruf eines mit <**Function**> definierten Funktionsblocks unter Beibehaltung des aktuellen Skript-Kontextes. Gibt den von der Funktion gelieferten Wert zurück. In KPath kann auch die alternative Syntax <***functionName***>**(...)** verwendet werden (<***functionName***> steht für den Namen der Funktion). Dies ist jedoch nur möglich, wenn der Funktionsname nicht gleich dem Namen einer KPath-Funktion ist.
Parameter:	*Beschreibung:*
@function	Name der Funktion. Alternativ kann auch mit folgender Syntax ein KPath-Ausdruck angegeben werden, der einen Funktionsnamen zurückgibt: ${KPath-Ausdruck}
<parameter>*	Aufrufparameter. Es wird der Rückgabewert der untergeordneten Anweisungen verwendet. Die Anzahl der Parameter muss mit der Anzahl der Argumente der Funktion übereinstimmen.

CallBehaviour **callBehaviour(selector, parameter1, ... parameterN)**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Aufruf einer Smalltalk-Methode, die bei der dem Topic zugeordneten Behaviour-Klasse implementiert ist.
Parameter:	*Beschreibung:*
@selector	Name der Methode
<parameter>*	Aufrufparameter analog zu **Call**

CallScript (ab K-Infinity 3.0) **callScript(script)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Aufruf eines registrierten Skripts im Ordner **[Systemordner** | **Registrierte Skripte]** unter Beibehaltung des aktuellen Skript-Kontextes. Angaben im <**script**>-Tag des aufgerufenen Skripts wie z.B. die Sprache werden dabei ignoriert.
Parameter:	*Beschreibung:*
@script	Name des Skripts.

CallScriptFunction (ab K-Infinity 3.0) **callScriptFunction(script, function, parameter1, ... parameterN)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Aufruf einer Funktion (s. **Call**) in einem registrierten Skript (s. **CallScript**)
Parameter:	*Beschreibung:*
@script	Name des Skripts.
@function	Name der Funktion. Alternativ kann auch mit folgender Syntax ein KPath-Ausdruck angegeben werden, der einen Funktionsnamen zurückgibt: ${KPath-Ausdruck}
<parameter>*	Aufrufparameter analog zu **Call**.

Function **-**	
Eingabe:	*Effekt/Ausgabe:*

-	Definiert einen Funktionsblock. Standardmäßig wird das Ergebnis der letzten Anweisung der Funktion an den Aufrufer zurückgeliefert. Mit <Return> kann aber auch explizit an beliebiger Stelle innerhalb der Funktion die Funktion beendet werden ein Wert zurückgegeben werden. Hinweis: Variablen in einer Funktion sollten immer deklariert werden, weil sie sonst implizit im Kontext des gesamten Skripts deklariert werden und damit seltsame Effekte auftreten können (z.B. bei Rekursion).
Parameter:	*Beschreibung:*
@name	Name der Funktion.
[@arguments]	Namen der Argumente, durch Kommata getrennt. Auf die Argumente kann in der Funktion wie auf lokale Variablen zugegriffen werden.

Beispiel (ohne Aufrufparameter):

```
<Script>
   <Function name="artists">
      <Path path="//'Künstler'/allInstances()/core()"/>
   </Function>

   <Call function="artists">
      <Each>
         <!- Name des Künstlers ausgeben ->
         <name/><cr/>
      </Each>
   </Call>
</Script>
```

Beispiel (mit Aufrufparameter):

```
<Function name="fakultaet" arguments="n">
   <If test="var(n) > 1">
      <do>
         <Return value="var(n) * fakultaet(var(n) - 1)"/>
      </do>
      <else>
         <Variable named="n"/>
      </else>
   </If>
</Function>

<Call function="fakultaet">
   <parameter><Path path="5"/></parameter>
   <do><value/></do>
</Call>
```

Import -	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Importiert alle Funktionen eines registrierten Skripts in ein aufrufendes Skript.
Parameter:	*Beschreibung:*
@script	Name des registrierten Skripts.

Return -	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Beendet eine Funktion und gibt optional das Ergebnis eines KPath-Ausdrucks zurück.
Parameter:	*Beschreibung:*
[@value]	KPath-Ausruck, der ausgwertet wird und als Ergebnis der Funktion zurückgegeben wird.

ThrowException -	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Generiert eine Fehlermeldung mit Rückgabewert
Parameter:	*Beschreibung:*
@path	KPath-Ausdruck, mit dem der Rückgabewert des Fehlers bestimmt wird
[@choose]	all, single, any, default-Wert all
[@showLocation]	Boolean. Wenn *true* wird Scriptname + Zeilennummer zur Fehlermeldung hinzugefügt
[@code]	HTTP-Fehlernummer für die Verwendung innerhalb eines REST-Reports

Try -	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Führt die in <do> eingeschlossenen Anweisungen aus. Mit den Anweisungen im <catch>-Block lassen sich Fehler, die z.B. per ThrowException ausgelöst werden, behandeln. Dazu muss im <catch>-Block der Parameter "parameter" auf den Namen der Umgebungsvariablen gesetzt werden.
Parameter:	*Beschreibung:*
<do>	Zu überwachender Block
<catch>	Fehlerbehandlungsblock, hierin Parameter "parameter" auf den Namen der Umgebungsvariablen setzen, in welche vor Ausführung des Fehlerbehandlungsblocks der Rückgabewert der Fehlermeldung geschrieben wird. Dann werden die Anweisungen in <catch> mit dem Rückgabewert als initiales Ergebnis ausgeführt

```
<Script>
  <Output>1,</Output>
  <Try>
    <do>
      <Output>2,</Output>
      <ThrowException path="'Hinweis,'"/>
      <Output>3,</Output>
    </do>
    <catch parameter="errorvar">
      <Output>4,</Output>
      <Path path="var(errorvar)"><value/></Path>
      <Output>5,</Output>
    </catch>
  </Try>
  <Output>6</Output>
</Script>
```

Ergibt die Ausgabe:

```
1,2,4,Hinweis,5,6
```

- **eval(kpath)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Führt die übergebene Zeichenkette als KPath Ausdruck aus.

Parameter:	*Beschreibung:*
@kpath	String, der den auszuführenden KPath Ausdruck enthält.

6.9 Bedingte Anweisungen

If **-**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Falls die Eingabe die angegebene Bedingung erfüllt wird die <**do**>- Anweisung ausgeführt, andernfalls die <**else**>-Anweisung
Parameter:	*Beschreibung:*
[@test]	KPath-Ausdruck, der einen Boolschen Wert zurückgibt. Zur Vereinfachung wird eine nichtleere Menge als **true** und eine leere Menge als **false** interpretiert.
[<condition>] (veraltet)	Bedingung (siehe Abschnitt „Bedingungen“). Statt <**condition**> sollte ein KPath-Ausdruck verwendet werden.
<do>	Anweisung, die bei erfüllter Bedingung ausgeführt wird.
[<else>]	Anweisung, die bei erfüllter Bedingung ausgeführt wird.

Beispiel:

```
<If test="@Alter >= 18">
   <do>
      <Output>Person ist volljährig</Output>
   </do>
   <else>
      <Output>Person ist nicht volljährig</Output>
   </else>
</If>
```

IfSize **-** (veraltet)

Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Wie bei <**If**>, wobei der Bedingungsteil die Größe der Eingabemenge testet.
Parameter:	*Beschreibung:*
@equal	Test auf |Objektmenge| = Parameter
@notEqual	Test auf |Objektmenge| != Parameter
@less	Test auf |Objektmenge| < Parameter
@lessOrEqual	Test auf |Objektmenge| <= Parameter
@greater	Test auf |Objektmenge| > Parameter
@greaterOrEqual	Test auf |Objektmenge| >= Parameter
<do>	Anweisung, die bei erfüllter Bedingung ausgeführt wird.
[<else>]	Anweisung, die bei erfüllter Bedingung ausgeführt wird.

Hinweis: <**IfSize**> kann durch <**If test="..."**> ersetzt werden. Aus <**IfSize greater="100"**> wird <**If test="size()** > **100"**>.

MaxDepth **-** (veraltet)	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Beschränkt die maximale Suchtiefe (Anzahl der Relationen von der Ausgangmenge des Skriptes). Unteranweisungen werden bei Überschreitung nicht ausgeführt.
Parameter:	*Beschreibung:*
@maxDepth	Maximale Suchtiefe.

MaxTime **-** (veraltet)	
Eingabe:	*Effekt/Ausgabe:*

beliebig	Beschränkt die Zeit, die zur Ausführung des Skriptes benötigt werden darf.
Parameter:	*Beschreibung:*
@maxTime	Maximale Zeit in Millisekunden.
@isGlobal	**true**/**false** [default: **false**]: **true**, falls bei Zeitüberschreitung die komplette Ausführung des Skriptes abgebrochen werden soll. Andernfalls wird nur die Ausführung der Unterelemente verhindert.

MaxTopics **-** (veraltet)	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Beschränkt die Anzahl der Topics, die vom Skript traversiert werden.
Parameter:	*Beschreibung:*
@maxTopics	Maximale Anzahl an Topics.
@isGlobal	siehe **MaxTime**.

Select **-** (veraltet)	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Wählt aus der Eingabemenge bestimmte Objekte aus.
Parameter:	*Beschreibung:*
<condition>	Bedingung (siehe Kapitel Bedingungen).
<do>	Anweisung, die mit den ausgewählten Objekte ausgeführt wird.

Hinweis: <**Select**> sollte durch einen entsprechenden <**Path**>-Ausdruck ersetzt werden.

While **-**

Eingabe:	*Effekt/Ausgabe:*
beliebig	Solange die angegebene Bedingung erfüllt ist, werden die innerhalb von <While> stehenden Anweisungen ausgeführt.
Parameter:	*Beschreibung:*
[@test]	KPath-Ausdruck, der einen Boolschen Wert zurückgibt. Zur Vereinfachung wird eine nichtleere Menge als **true** und eine leere Menge als **false** interpretiert.

6.10 Mengen

Add **add(value)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Fügt das über <***value***> ermittelte Objekt zur Eingabemenge hinzu.
@value	KPath-Ausdruck, liefert das Objekt, welches zur Mengen hinzugefügt werden soll

AnyElement **anyElement()**	
Alternativer Name:	AnyValue, anyValue()
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Falls genau ein beliebiges Element der Menge zurück. Ist die Menge leer, wird nichts geliefert.
Einzelnes Objekt	Das Objekt.

AtIndex **atIndex(index)**	
Eingabe:	*Effekt/Ausgabe:*

Geordnete Menge von Objekten	Führt die untergeordneten Anweisungen mit dem Objekt der Menge aus, das an Position <***index***> steht. Das erste Element hat die Position 1.
Nicht geordnete Mengen oder andere Objekte	Kein Rückgabewert.
Parameter:	*Beschreibung:*
@index	KPath-Ausdruck, liefert die Position des ausgewählten Objektes.

AtIndexPut **atIndexPut(index,value)**	
Eingabe:	*Effekt/Ausgabe:*
Geordnete Menge von Objekten	Setzt den Eintrag an der über <***index***> angegebenen Position auf den über <***value***> angegebenen Wert und liefert das Eingabeobjekt zurück.
Nicht geordnete Mengen oder andere Objekte	Kein Rückgabewert.
Parameter:	*Beschreibung:*
@index	KPath-Ausdruck, liefert die Position des Listeneintrags
@value	KPath-Ausdruck, liefert den zukünftigen Wert des Listeneintrags

AtKey **atKey(key)**	
Eingabe:	*Effekt/Ausgabe:*
Schlüssel/Wert-Paare	Liefert den Wert, der unter dem Schlüssel gespeichert ist, oder **null**.
Parameter:	*Beschreibung:*
@key	Schlüssel

AtKeyPut **atKeyPut(key, value)**	
Eingabe:	*Effekt/Ausgabe:*
Schlüssel/Wert-Paare	Setzt den Wert, der unter dem Schlüssel gespeichert ist
Parameter:	*Beschreibung:*
@key	Schlüssel
@value	Wert

Break **-**	
Eingabe:	*Effekt/Ausgabe:*
-	Bricht die Iteration einer Menge ab (s. <**Each**>).

Collect **collect(collect)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Führt einen Ausdruck für jedes der Eingabeobjekte einzeln aus und liefert die Menge der Einzelergebnisse zurück.
Parameter:	*Beschreibung:*
<collect>	Der je Element auszuwertende Ausdruck
<do>	Die gesammelten Ausdrücke bilden die Eingabe des do-Blocks

Each **-**	
Alternativnamen:	Do, ForEach, TraverseEach
Eingabe:	*Effekt/Ausgabe:*

Menge von Objekten	Iteriert die Menge und führt die untergeordneten Anweisungen mit jedem einzelnen Objekt der Menge aus.
Einzelnes Objekt	Führt die untergeordneten Anweisungen mit dem Objekt aus.

Beispiel:

```
<Attributes>
   <Each>
      <!- einzelnes Attribut ->
   </Each>
</Attributes>
```

Flatten **flatten()**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Liefert eine Liste aller einzelnen Elemente der Menge. Geschachtelte Mengen werden rekursiv iteriert und die enthaltenen Elemente ebenfalls hinzugefügt.

FindMax **findMax(path, [type])**	
Eingabe:	*Effekt/Ausgabe:*
Menge	Bestimmt aus der Menge dasjenige Element, welches für die Auswertung mit dem angegeben KPath-Ausdruck @path das Maximum ergibt
Parameter:	*Beschreibung:*
@path	benötigter Parameter: der KPath-Ausdruck, der für jedes Element der Menge ausgewertet wird
[@type]	@type gibt an, ob die resultierenden Objekte (type="objects") oder ihre Werte (type="literals") verglichen werden. Standard ist "literals"

FindMaxValue **findMaxValue(path, [type])**

Eingabe:	*Effekt/Ausgabe:*
Menge	Bestimmt aus der Menge denjenigen Wert, welcher für die Auswertung der einzelnen Elemente mit dem angegeben KPath-Ausdruck @path das Maximum ergibt
Parameter:	*Beschreibung:*
@path	benötigter Parameter: der KPath-Ausdruck, der für jedes Element der Menge ausgewertet wird
[@type]	@type gibt an, ob die resultierenden Objekte (type="objects") oder ihre Werte (type="literals") verglichen werden. Standard ist "literals"

FindMin **findMin(path, [type])**	
Eingabe:	*Effekt/Ausgabe:*
Menge	Bestimmt aus der Menge dasjenige Element, welches für die Auswertung mit dem angegeben KPath-Ausdruck @path das Minimum ergibt
Parameter:	*Beschreibung:*
@path	benötigter Parameter: der KPath-Ausdruck, der für jedes Element der Menge ausgewertet wird
[@type]	@type gibt an, ob die resultierenden Objekte (type="objects") oder ihre Werte (type="literals") verglichen werden. Standard ist "literals"

FindMinValue **findMinValue(path, [type])**	
Eingabe:	*Effekt/Ausgabe:*
Menge	Bestimmt aus der Menge denjenigen Wert, welcher für die Auswertung der einzelnen Elemente mit dem angegeben KPath-Ausdruck @path das Minimum ergibt
Parameter:	*Beschreibung:*
@path	benötigter Parameter: der KPath-Ausdruck, der für jedes Element der Menge ausgewertet wird
[@type]	@type gibt an, ob die resultierenden Objekte (type="objects") oder ihre Werte (type="literals") verglichen werden. Standard ist "literals"

Groups **-** (veraltet)	
Eingabe:	*Effekt/Ausgabe:*
Menge von Individuen, Relationen oder Attributen	Gruppiert die Objekte nach ihrem Prototyp und führt untergeordnete Anweisungen mit den einzelnen Gruppen aus.

ObjectsOfGroup **objectsOfGroup()** (veraltet)	
Alternativnamen:	TopicsOfGroup
Eingabe:	*Effekt/Ausgabe:*
Gruppe	Klammert die Objekte der Gruppe und liefert sie als Resultat zurück

GroupType **-** (veraltet)	
Eingabe:	*Effekt/Ausgabe:*
Gruppe	Prototyp der Gruppe.

Beispiel:

```
<Attributes>
   <Groups>
      <Each>
         <GroupType>
            <Output>Attribut: <name/><cr/></Output>
         </GroupType>
         <Output>Werte: </Output>
         <Each>
```

```
            <Output><value/></Output>
          </Each>
          <cr/>
        </Each>
      </Groups>
    </Attributes>
```

Ausgabe:

```
Attribut: Name
Werte: FAQ
Attribut: Änderungsdatum
Werte: 22.3.2002 13.5.2002 18.9.2002
```

GroupBy **groupBy(groupBy)**	
Alternativnamen:	TraverseGroups,Groups
Eingabe:	*Effekt/Ausgabe:*
Menge	Für die Elemente werden Schlüssel/Wert-Paare erzeugt. Der Schlüssel wird ausgehend von den Elementen berechnet. Unter einem Schlüssel werden alle Elemente gespeichert, die denselben Schlüssel haben, d.h. der Wert ist eine Teilmenge der Elemente.
Parameter:	*Beschreibung:*
@groupBy	benötigter Parameter: KPath, der ausgehend von einem Element den Schlüssel berechnet.

GroupKey **groupKey()**	
Alternativnamen:	GroupType
Eingabe:	*Effekt/Ausgabe:*
Gruppe	Liefert den Schlüssel eines Schlüssel/Wert-Paares

GroupValue **groupValue()**	
Eingabe:	*Effekt/Ausgabe:*

Gruppe	Liefert den Wert eines Schlüsse/Wert-Paares

Intersection **intersection(path*1*, ... , path*N*)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Schneidet die Eingabemenge mit anderen Mengen.
Parameter:	*Beschreibung:*
[@includeInput]	**true**/**false** [default: **true**]: **false**, wenn die Eingabemenge nicht mit in die Schnittmenge einbezogen werden soll. Siehe Beispiel bei **Union**. In KPath kann dieser Parameter nicht angegeben werden.
<with>*	Menge, mit der die Eingabe geschnitten wird.
<do>	Anweisung, die mit der Schnittmenge als Eingabe ausgeführt wird.

Beispiel 1: finde alle Personen, deren Name mit "G" beginnt und die Anwalt sind.

```
intersection(//Person\G*, //Anwalt/instances()/core())
```

Beispiel 2: alle Künstler iterieren, deren Name mit "B" beginnt

```
<Path path="//Person\B*">
   <Intersection>
      <with>
         <Path path=‘‘//Beruf\Künstler\*/core()‘‘/>
      </with>
      <do>
         ...
      </do>
   </Intersection>
</Path>
```

Beispiel 3: die Menge alle Bücher und Personen

```
<SetVariable variable="test">
   <Path path="//Buch\*"/>
</SetVariable>
<Union includeInput="false">
   <with>
      <Path path="var(test)"/>
```

```
    </with>
    <with>
       <Path path="//Person\*"><Core/></Path>
    </with>
    <do><Output/></do>
</Union>
```

- **list(path*1*, ..., path*N*)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Liefert für jedes Objekt eine Liste, deren Elemente durch Auswertung der KPath-Ausdrücke entstehen.
Parameter:	*Beschreibung:*
@path*X*	KPath-Ausdruck, der jeweils ein Element der Liste liefert.

Beispiel: Name und Geburtstag aller Personen

```
//Person/instances()/list(name(), @Geburtstag/value())
```

MaxDepth **maxDepth(maxDepth)** veraltet	
Eingabe:	*Effekt/Ausgabe:*
-	Bricht die Bearbeitung bei Erreichen der Schachtelungstiefe @maxDepth ab, Rückgabewert null.
Parameter:	*Beschreibung:*
@maxDepth	maximale Schachtelungstiefe

NewList **newList()**	
Alternativnamen:	EmptyList
Eingabe:	*Effekt/Ausgabe:*

-	Neue (leere) Liste (Menge in geordneter Reihenfolge)

NewMap **newMap()**	
Alternativnamen:	EmptyGroup
Eingabe:	*Effekt/Ausgabe:*
-	leere Menge von Schlüssel/Wert-Paaren

Remove **remove(value)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Entfernt ein über <***value***> ermitteltes Objekt aus der Eingabemenge und liefert diese zurück.
@value	KPath-Ausdruck, liefert das Objekt, welches aus der Menge entfernt werden soll

RemoveAll **removeAll(value)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Entfernt alle Vorkommen des über <***value***> ermittelten Objektes aus der Eingabemenge und liefert diese zurück.
@value	KPath-Ausdruck, liefert das Objekt, welches aus der Menge entfernt werden soll

Size **size()**

Alternativnamen:	Cardinality, cardinality
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Größe der Menge, Ganzzahl
Einzelnes Objekt	1

Subtract **subtract(path*1*, ..., path*N*)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Abzugsoperation. Von der Eingangsmenge werden alle folgenden ermittelten Mengen abgezogen.
Parameter:	*Beschreibung:*
[@includeInput]	**true**/**false** [default: **true**]: **false**, wenn die Eingabemenge nicht mit einbezogen werden soll. Siehe Beispiel bei **Union**. In KPath kann dieser Parameter nicht angegeben werden.
<with>*	Zu verarbeitende Mengen.
<do>	Anweisung, die mit der Abzugsmenge als Eingabe ausgeführt wird.

Sublist **sublist(from, to, [checkBounds])**	
Eingabe:	*Effekt/Ausgabe:*
Menge	Bestimmt für die KPath-Ausdrücke @from und @to Indexwerte und gibt die Untermenge von @from bis @to zurück
Parameter:	*Beschreibung:*
@from	KPath-Ausdruck für den Startindex
@to	KPath-Ausdruck für den Stoppindex
@checkBounds	Boolean, welcher angibt, ob die berechneten Indexe valide Werte innerhalb der Menge bilden müssen oder ob auf Feldanfang bzw. -ende ausgewichen werden darf. default-Wert false

Sort **sort(path, [ascending], [type])**	
Eingabe:	*Effekt/Ausgabe:*
Topics, Attribute, Relationen	Sortiert die Eingabeobjekte nach einem KPath-Ausdruck.
Parameter:	*Beschreibung:*
@path	Sortierungs-Kriterium (KPath-Ausdruck).
[@ascending]	**true**/**false** [default: **true**]: • **true**: aufsteigende Sortierung • **false**: absteigende Sortierung
[@type]	**literals**/**objects** [default: **literals**]: • **literals**: vergleicht auf Basis des Wertes des Objektes • **objects**: verwendet die bei den Wissensnetzobjekten definierten Vergleichsmethoden (z.B. für Datumsattribute)

Beispiel:

```
<Sort path="@Name" ascending="false">
    <do>
       <Output>Name:<name/></Output>
    </do>
 </Sort>
```

Will man über mehrere Eigenschaften geschachtelt sortieren, so lässt sich dies über die zusätzliche Angabe der gewünschten Eigenschaften im Tag <sortKey> erreichen.

Beispiel:

```
<Sort path="@Name" ascending="false">
   <sortKey path="@shoesize/value()" ascending="true" type="literals"/>
   <sortKey path="@birthday/value()" ascending="true" type="literals"/>

   <do>
      <Output>Name:<name/></Output>
   </do>
</Sort>
```

Hier wird zunächst nach dem Namen, bei Gleichheit nach der Schuhgröße, bei dortiger Gleichheit nach dem Geburtstag sortiert. Existiert der Wert für die Sortierung nicht, so wird das Objekt auf der jeweiligen Stufe zuletzt eingeordnet. Achtung: Das sortKey-Tag steht

z.Zt. nur in KScript zur Verfügung!

Union **union(path*1*, ..., path*N*)**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Vereinigt die Eingabemenge mit anderen Mengen.
Parameter:	*Beschreibung:*
[@includeInput]	**true**/**false** [default: **true**]: **false**, wenn die Eingabemenge nicht mit in die Vereinigungsmenge einbezogen werden soll. In KPath kann dieser Parameter nicht angegeben werden.
<with>*	Menge(n), mit der die Eingabe vereinigt wird.
<do>	Anweisung, die mit der Vereinigungsmenge als Eingabe ausgeführt wird.

Beispiel: alle Personen iterieren, deren Namen mit "A" oder "B" beginnt

```
<Path path="//Person">
   <Union includeInput="false">
      <with>
         <Path path="\A*"/>
      </with>
      <with>
         <Path path="\B*"/>
      </with>
      <do>
         ...
      </do>
   </Union>
</Path>
```

WithoutDuplicates **withoutDuplicates([path])**	
Alternativnamen:	AsSet, asSet()
Eingabe:	*Effekt/Ausgabe:*
Menge von Objekten	Liefert die Menge ohne Mehrfachvorkommen von einzelnen Elementen.
Einzelnes Objekt	Liefert eine Menge mit dem Objekt.

Parameter:	*Beschreibung:*
[@path]	Benutzt die durch **path** beschriebene Menge, die statt der Eingabe verwendet wird.

6.11 Zeichenketten und reguläre Ausdrücke

AnalyzeString **-**	
Eingabe:	*Effekt/Ausgabe:*
eine oder mehrere Zeichenketten	Sucht in der Zeichenkette alle Teilzeichenketten, die einem regulären Ausdruck entsprechen, und führt mit diesen jeweils die Anweisung im Unterelement <**matchingSubstring**> aus. Auf die entsprechende Teilzeichenkette kann mit **regexMatch()** zugegriffen werden. Auf den Teil, der einer Gruppe N (n >= 1) innerhalb des regulären Ausdrucks entspricht, kann mit **regexGroup(n)** zugegriffen werden. **regexGroup(0)** entspricht **regexMatch()**. Für Teilzeichenketten, die nicht dem regulären Ausdruck entsprechen, wird das Unterelement <**nonMatchingSubstring**> ausgeführt. Auf die nicht dem regulären Ausdruck entsprechende Teilzeichenkette kann mit **regexNonMatch()** zugegriffen werden.
Parameter:	*Beschreibung:*
@regex	Regulärer Ausdruck. Alternativ kann auch mit folgender Syntax ein KPath-Ausdruck angegeben werden, der einen regulären Ausdruck zurückgibt: ${KPath-Ausdruck}
[@string]	KPath-Ausdruck, auf den statt auf die Eingabe der reguläre Ausdruck angewendet wird.

Beispiel: Zerlegung einer Zeichenkette, die Postleitzahlen und Städtenamen enthält

```
<AnalyzeString
      string="'+++ 64283 Darmstadt ###  10785 Berlin /'"
      regex="([0-9]+)\s*([a-zA-Z]+)">
   <matchingSubstring>
      <Output>Match: </Output>
      <path path="regexMatch()"/><cr/>
      <Output>PLZ: </Output>
      <path path="regexGroup(1)"/><cr/>
      <Output>Stadt: </Output>
      <path path="regexGroup(2)"/><cr/>
```

```
    </matchingSubstring>
    <nonMatchingSubstring>
       <Output>Non-match: </Output>
       <path path="regexNonMatch()"/><cr/>
    </nonMatchingSubstring>
 </AnalyzeString>
```

Ausgabe:

```
Non-match: +++
Match: 64283 Darmstadt
PLZ: 64283
Stadt: Darmstadt
Non-match:  ###
Match: 10785 Berlin
PLZ: 10785
Stadt: Berlin
Non-match:  /
```

AsLowercase **asLowercase()**	
Eingabe:	*Effekt/Ausgabe:*
Zeichenkette	Wandelt alle Buchstaben in dem String in Kleinbuchstaben um.

AsUppercase **asUppercase()**	
Eingabe:	*Effekt/Ausgabe:*
Zeichenkette	Wandelt alle Buchstaben in dem String in Großbuchstaben um.

- **concat(separator, path*1*, ..., path*N*)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Die Ergebnisse der Path-Ausdrücke **path*1*** bis **path*N***, angewendet auf das oder die Zielobjekt/e werden durch **separator** getrennt aneinandergefügt und die zusammengesetzte Zeichenkette zurückgegeben.
Parameter:	*Beschreibung:*

@separator	Zeichenkette zur Trennung der Ergebnisse der einzelnen KPath-Ausdrücke.
@path*X*	KPath-Ausdruck.

Beispiel:

```
//Person\Goethe/concat(",", name(), @birthday/value(), @address/value())
```

Ergebnis:

```
Goethe,28.08.1749,Goethehaus Frankfurt Ffm
```

- **digest([hash], [encoding])**	
Eingabe:	*Effekt/Ausgabe:*
Zeichenkette	Hashwert der Zeichenkette
Parameter:	*Beschreibung:*
@hash	'MD5', 'SHA-1' oder 'SHA-256'. Standard ist 'MD5'
@encoding	Darstellung des Hashwerts. Entweder 'hex' oder 'base64'. Standard ist 'hex'.

- **expand(target, arg*1*, ... , arg*N*)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Der Wert des KPath-Ausdrucks **target** mit den eingesetzten Werten der Ausdrücke **arg*1*** ... **arg*N*** für die Teilstrings <*1s*> ... <*Ns*>.
Parameter:	*Beschreibung:*
@target	KPath-Ausdruck, der eine Zeichenkette liefert mit eingebetteten "<*X*s>" Teilzeichenketten. Die Syntax für die Ersetzungsausdrücke ist identisch mit der für virtuelle Tabellenspalten im Tabellenimport (s. Handbuch zum Knowledge Builder - "Syntax für Virtuelle Tabellenspalten")
@arg*X*	KPath-Ausdrücke, die für die "<*X*s>" Ausdrücke in **target** ersetzt werden.

Beispiel:

```
expand("<2s><1s><1s><1s> erfolgreich<1s>", ".", "Test")
```

Ergebnis:

```
Test... erfolgreich.
```

Length **length()**	
Eingabe:	*Effekt/Ausgabe:*
Zeichenkette	Länge der Zeichenkette.

MatchesRegEx **matchesRegEx(regex)**	
Alternativname:	MatchesRegex, matchesRegex()
Eingabe:	*Effekt/Ausgabe:*
Zeichenkette	Ergibt **true** wenn die Zeichenkette dem regulären Ausdruck entspricht, sonst **false**.
Parameter:	*Beschreibung:*
@regex	Regulärer Ausdruck. Anstelle eines regulären Ausdrucks kann auch ein KPath-Ausdruck angegeben werden, der eine Zeichenkette mit einem regulären Ausdruck zurückliefert (z.B. kann der reguläre Ausdruck auch ein Attributwert sein, der vom KPath-Ausdruck zurückgeliefert wird).

ReplaceAll **replaceAll(substring, replacement)**	
Eingabe:	*Effekt/Ausgabe:*
Zeichenkette	Ersetzt alle Vorkommen von **substring** durch **replacement**.
Parameter:	*Beschreibung:*
@substring	Zu ersetzende Zeichenkette.

@replacement	Ersatzzeichenkette.

- **splitString(separators [, string])**	
Eingabe:	*Effekt/Ausgabe:*
Zeichenkette	Liefert eine Menge von Teilzeichenketten. Eine oder mehrere Zeichenketten, angegeben entweder durch den Path-Ausdruck **string** oder durch das übergeordnete Objekt, werden mit Hilfe der Zerlegungszeichen **separators** zerlegt. Anmerkung: Leere gefundene Teilzeichenketten werden nicht entfernt.
Parameter:	*Beschreibung:*
@separators	Ein oder mehrere Zeichen. Diese werden als Trenner angesehen.
[@string]	Zeichenkette, die zerlegt werden soll.

Beispielaufrufe:

```
<Path path="splitString(',+','habel, gabel und + weiteres')">
    <Do><path path="."/><cr/></Do>
</Path>
<Path path="splitString('a',//Branche\*/name())">
    <Do><path path="."/><cr/></Do>
</Path>
<Path path="//Branche\*/name()/splitString('a')">
    <Do><path path="."/></Do>
</Path>
```

Substring **substring(start, length)**	
Eingabe:	*Effekt/Ausgabe:*
Zeichenkette	Liefert eine Teilzeichenkette der Eingabe die bei **start** anfängt und sich über **length** Zeichen erstreckt.
Parameter:	*Beschreibung:*
@start	Position an der die Teilzeichenkette beginnen soll.

[@length]	Länge der Teilzeichenkette (optional). Wir dieser Parameter nicht angegeben, so werden alle Zeichen von "start" bis zum Ende der Zeichenkette zurückgeliefert.

6.12 Zahlen

AsNumber **asNumber()**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Zeichenketten	Interpretiert jede Zeichenkette als Zahl und liefert eine Menge von Zahlen zurück.
Beispiel	KPath: union('1','2','3')/asNumber() - 1 Ausgabe: 0,1,2

Average **average()**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Zahlen	Liefert den numerischen Durchschnitt der Menge der Zahlen. Ist die Menge leer, wird 0 zurück geliefert.

Ceiling **ceiling()**	
Eingabe:	*Effekt/Ausgabe:*
Zahl	Wandelt Zahlen in Ganzzahlen um, wobei der nächst größere Wert gewählt wird, wenn die Eingabezahl nicht selbst als Ganzzahl darstellbar ist, also wird 3.141592 zur 4 und 3.00000 zu 3.

Floor **floor()**	
Eingabe:	*Effekt/Ausgabe:*
Zahl	Integerwert der Zahl ohne Nachkommastellen

LowerMedian **lowerMedian()**	
Eingabe:	*Effekt/Ausgabe:*
Zahlenfeld	gibt den Wert an der mittleren Position im Zahlenfeld an, bei ungerader Anzahl den Wert links von der Mittelposition

Product **product()**	
Eingabe:	*Effekt/Ausgabe:*
Zahlenfeld	Gibt das Produkt aller Zahlen im Zahlenfeld zurück

RaisedTo **raisedTo(power)**	
Eingabe:	*Effekt/Ausgabe:*
Zahl/Zahlenfeld	Berechnet die Potenz der Zahl/Zahlen und zwar jeweils hoch @power
Parameter:	*Beschreibung:*
@power	Zahl oder KPath-Ausdruck, der die Potenz berechnet

Rounded **rounded()**	
Eingabe:	*Effekt/Ausgabe:*

Zahl/Zahlenfeld	Rundet die Zahl/die Elemente des Zahlenfelds

Sqrt **sqrt()**	
Eingabe:	*Effekt/Ausgabe:*
Zahl/Zahlenfeld	Berechnet für die Zahl/die Elemente des Zahlenfeldes die Wurzel und gibt diese zurück

StandardDeviation **standardDeviation()**	
Eingabe:	*Effekt/Ausgabe:*
Zahlenfeld	Berechnet für die Elemente des Zahlenfeldes die Standardabweichung und gibt diese zurück

Sum **sum()**	
Eingabe:	*Effekt/Ausgabe:*
Zahlenfeld	Liefert die Summe der Zahlen zurück

SumBy **sumBy(sumBy)**	
Eingabe:	*Effekt/Ausgabe:*
Menge	Elemente werden iteriert, das jeweilige Ergebnis des Pfades geht in die Summe ein
Parameter:	*Beschreibung:*

@sumBy	benötigter Parameter: KPath, der für jedes Element ausgewertet wird und für die Summierung angewendet wird

UpperMedian **upperMedian()**	
Eingabe:	*Effekt/Ausgabe:*
Zahlenfeld	gibt den Wert an der mittleren Position im Zahlenfeld an, bei ungerader Anzahl den Wert rechts von der Mittelposition

6.13 Datum und Uhrzeit

AsLocalTimestampFromUTC **asLocalTimestampFromUTC()**	
Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Konvertiert eine UTC Zeitangabe in eine lokale Zeitangabe

AsMilliseconds **asMilliseconds()**	
Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Wandelt die Zeitangabe in die Anzahl der Millisekunden seit 1.1.1901 um.

AsSeconds **asSeconds()**	
Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Wandelt die Zeitangabe in die Anzahl der Sekunden seit 1.1.1901 um.

AsTimestampForUTC **asTimestampForUTC()**	
Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Konvertiert eine lokale Zeitangabe in eine UTC Zeitangabe

CurrentDate **currentDate()**	
Eingabe:	*Effekt/Ausgabe:*
-	Ergibt den Datumswert des aktuellen Datums auf dem ausführenden Computer.

CurrentTime **currentTime()**	
Eingabe:	*Effekt/Ausgabe:*
-	Ergibt den Zeitwert der aktuellen Uhrzeit auf dem ausführenden Computer.

CurrentTimestamp **currentTimestamp()**	
Eingabe:	*Effekt/Ausgabe:*
-	Ergibt einen Zeitstempel (Datum und Uhrzeit) der Uhrzeit auf dem ausführenden Computer.

Date **date(dayInMonth, month, year)**	
Eingabe:	*Effekt/Ausgabe:*

beliebig	Gibt ein Datum zurück, um z.B. Vergleiche mit Datumsattributen durchzuführen. Bei Angabe von dayInMonth (Tag), month (Monat) und year (Jahr) wird das angegebene Datum zurückgegeben. Bei der Jahreszahl wird keine Spezialbehandlung durchgeführt, d.h. "73" wird als Jahr 73 direkt umgesetzt, und nicht als 1973. Wenn dayInMonth, month und year nicht angegeben werden, und die Eingabe ein Attribut ist, wird der Wert als Datum zurückgegeben, falls möglich. Bei flexiblen Zeitattributen (ab K-Infinity 3.0) müssen die Datumsangaben komplett sein. Andernfalls wird das aktuelle Datum zurückgeben.
Parameter:	*Beschreibung:*
[@dayInMonth]	KPath, der den Tag des Monats (1-(28-31)) bestimmt
[@month]	KPath, der die Nummer des Monats (1-12) bestimmt
[@year]	KPath, der das Jahr bestimmt

DateAsDays **dateAsDays()**	
Eingabe:	*Effekt/Ausgabe:*
Datum	Anzahl der Tage seit Startsystemzeit bis zum angegebenen Datum

DateString **dateString()**	
Eingabe:	*Effekt/Ausgabe:*
Datumsattribut	Gibt den Attributwert als Datumsangabe in Form einer Zeichenkette zurück, falls möglich. Bei flexiblen Zeitattributen (ab K-Infinity 3.0) werden auch unvollständige Datumsangaben wie z.B. "1984" zurückgegeben.

DayOfMonth **dayOfMonth()** veraltet	
Eingabe:	*Effekt/Ausgabe:*

Datums- bzw. Zeitangabe	Gibt den Tag (1-31) der Zeitangabe zurück

DayOfMonthValue **dayOfMonthValue()**	
Eingabe:	*Effekt/Ausgabe:*
Datums- bzw. Zeitangabe	Gibt den Tag (1-31) der Zeitangabe zurück

DaysInMonth **daysInMonth()**	
Eingabe:	*Effekt/Ausgabe:*
Datums- bzw. Zeitangabe	Anzahl der Tage in dem angegebenen Monat

FormatTimestamp **formatTimestamp(format, [language])**	
Eingabe:	*Effekt/Ausgabe:*
Timestamp	Gibt Timestamp in einem Format aus, welches gemäß der Auswertung des @format-KPaths zusammengestellt ist
Parameter:	*Beschreibung:*
@format	KPath-Ausdruck, der eine gültige Formatangabe ergeben muss
[@language]	Sprachangabe, z.B. für die Ausgabe von Monatsnamen relevant

Zusammenstellung der @format-Bestandteile:

TAB,CR,SPACE	erlaubtes Whitespace
''	unbehandelter Text in Anführungszeichen geklammert

-./:	Separatoren
?	zur Kennzeichnung optionaler Elemente in der Timestamp (angehängt)
AMPM	am oder pm Marker
AZ	Buchstaben
D	Tag (Zahl, 1-31, 1 oder 2 stellig)
DD	Tag (Zahl, 01-31, 2 stellig)
h	Stunde (Zahl 0-23, 1 oder 2 stellig)
hh	Stunde (Zahl 00-23, 2 stellig)
m	Minute (Zahl, 1 oder 2 stellig)
mm	Minute (Zahl, 2 stellig)
M	Monat (Zahl, 1 oder 2 stellig)
MM	Monat (Zahl, 2 stellig)
MMM	Monat (String, 3 stellig, sprachabhängig)
mmm	Millisekunden (Zahl, 3 stellig)
MMMM	Monat (String, sprachabhängig)
nn*	Sub-Sekunden (Zahl, beliebige Stellenzahl)
s	Sekunden (Zahl, 1 oder 2 stellig)
ss	Sekunden (Zahl, 1 oder 2 stellig)
YY	Jahr (Zahl, 2 stellig)
YYYY	Jahr (Zahl, 4 stellig)

Month **month()** veraltet	
Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Nummer des Monats, in dem die Zeitangabe liegt

MonthValue **monthValue()**

Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Nummer des Monats, in dem die Zeitangabe liegt

ParseTimestamp **parseTimestamp(format, [language])**	
Eingabe:	*Effekt/Ausgabe:*
String	Parsed die Eingabe mit dem angegebenen Format und gibt einen Timestamp zurück.
Parameter:	*Beschreibung:*
@format	KPath-Ausdruck, der eine gültige Formatangabe ergeben muss (siehe FormatTimestamp)
[@language]	Sprachangabe, z.B. für die Ausgabe von Monatsnamen relevant

Time **time([hour], [minute], [second])**	
Eingabe:	*Effekt/Ausgabe:*
Objekt	Ergibt einen Datumwert mit den angegebenen Werten. Wenn mind. einer der optionalen Werte fehlt, wird zunächst versucht aus dem übergebenen Objekt einen Zeitwert zu bestimmen, falls das aus dem Objekt nicht möglich ist, wird die aktuelle Uhrzeit zurückgeliefert.
Parameter:	*Beschreibung:*
[@hour]	KPath, der den Stundenwert bestimmt (0-23)
[@minute]	KPath, der den Minutenwert bestimmt (0-59)
[@second]	KPath, der den Sekundenwert bestimmt (0-59)

Timestamp **timestamp(([dayInMonth], [month], [year],[hour], [minute], [second])**	
Eingabe:	*Effekt/Ausgabe:*

Objekt	Ergibt einen Zeitstempel mit den angegebenen Werten. Wenn mind. einer der optionalen Werte fehlt, wird zunächst versucht aus dem übergebenen Objekt einen Zeitstempel zu bestimmen, falls das aus dem Objekt nicht möglich ist, wird das aktuelle Datum und die aktuelle Uhrzeit zurückgeliefert.
Parameter:	*Beschreibung:*
[@dayInMonth]	KPath, der den Tag des Monats (1-(28-31)) bestimmt
[@month]	KPath, der die Nummer des Monats (1-12) bestimmt
[@year]	KPath, der das Jahr bestimmt
[@hour]	KPath, der den Stundenwert bestimmt (0-23)
[@minute]	KPath, der den Minutewert bestimmt (0-59)
[@second]	KPath, der den Sekundenwert bestimmt (0-59)

Year **year()** veraltet	
Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Jahr, in dem die Zeitangabe liegt

YearValue **yearValue()**	
Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Jahr, in dem die Zeitangabe liegt

WeekdayIndex **weekdayIndex()**	
Eingabe:	*Effekt/Ausgabe:*
Zeitangabe	Nummer des Tages einer Woche (1:Montag, ..., 7:Sonntag)

6.14 Suchen

ExpertQuery **-**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Sucht mit Hilfe der angegebenen Expertensuche nach dem zu berechnenden Suchstring, Ergebnis sind Hits.
Parameter:	*Beschreibung:*
@queryName	Externe ID der auszuführenden Expertensuche (als Zeichenkette oder als in ${...} gekapselter KPath-Ausdruck)
[@filterInput]	Wenn *true* wird die Expertensuche als Filter für die Eingabe verwendet
<parameter>*	Aufzählung von name/value-Paaren, die als Parameter in die Expertensuche gegeben werden

Beispiel:

```
<ExpertQuery queryName="findTopicWithID">
    <parameter name="myID" value="var(id)"/>
    <Each>
        <Output>Name: <name/></Output><cr/>
    </Each>
</ExpertQuery>
```

Dieses Script führt die Expertensuche 'findTopicWithID' mit dem Inhalt der Variablen 'id' als Parameter aus und gibt die Namen der Treffer aus.

SimpleSearch **simpleSearch(searchName,searchString)**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Baut eine einfache Suche gemäß der Suchkonfiguration mit Namen @searchName und führt sie mit dem Suchstring @searchString aus. Ergebnis ist die Treffermenge
Parameter:	*Beschreibung:*
@searchName	Namen der Suchkonfiguration (als Zeichenkette oder als in ${...} gekapselter KPath-Ausdruck)
@searchString	Zu suchende Zeichenkette (als Zeichenkette oder als in ${...} gekapselter KPath-Ausdruck)

Beispiel:

```
<Path path="//$person$/instances()">
    <Each>
        <Output>Trigram-Varianten von <name/>:</Output><cr/>
        <SimpleSearch searchName="trigram" searchString="${name()}">
            <Each>
                <Output>    </Output><name/><cr/>
            </Each>
        </SimpleSearch>
    </Each>
</Path>
```

Die Suche 'trigram' wird mit dem Namen jeder Person als zusuchender Zeichenkette ausgeführt und die Suchergebnisse ausgegeben.

SimpleSearchConfig **-**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt die Suchkonfiguration mit angegebenem Namen zurück. Dient der Schemabearbeitung und -abfrage.
Parameter:	*Beschreibung:*
@searchName	Name der Such-Konfiguration. Es sind entweder die aktiv gesetzte Suchbeschreibung oder der interne Name benutzbar.

SetHitQuality **setHitQuality(quality)**	
Eingabe:	*Effekt/Ausgabe:*
Suchtreffer	Setzt die Qualität des Suchtreffers auf den Wert, der sich aus dem KPath-Ausdruck @quality ergibt
Parameter:	*Beschreibung:*
@quality	KPath-Ausdruck zur Festlegung der Qualität

Causes **causes()**	
Eingabe:	*Effekt/Ausgabe:*
Menge von Suchtreffern	Liefert für jeden Suchtreffer die Treffergründe (causes).

Property **property(propertyName)**	
Eingabe:	*Effekt/Ausgabe:*
Suchtreffer/Cause	Liefert den Wert der Eigenschaft @propertyName. Falls ein Hit übergeben wird, wird eine Menger der Eigenschafter aller Cause geliefert.
Parameter:	*Beschreibung:*
@propertyName	Name der Eigenschaft

SetProperty **setProperty(propertyName, value)**	
Eingabe:	*Effekt/Ausgabe:*
Suchtreffer/Custom cause	Setzt den Wert der Hit-Eigenschaft @propertyName mit dem Wert des KPath-Ausdrucks @value. Es wird dazu, falls noch nicht vorhanden, ein Cause vom Typ "custom" angelegt
Parameter:	*Beschreibung:*
@propertyName	Name der Eigenschaft
@value	Durch KPath bestimmter Wert der Eigenschaft

CustomCause **customCause()**	
Eingabe:	*Effekt/Ausgabe:*

Suchtreffer	Liefert einen Cause vom Typ "custom" der Eigenschaft. Bei diesem können dann die mit setProperty gesetzen Eigenschaften abgefragt werden, z.B. customCause()/property(”factor”)
Parameter:	*Beschreibung:*
[@modifyExisting]	True: Wenn genau ein Cause bereits vorhanden ist, wird dieser zurückgegeben. In allen anderen Fällen wird ein neuer Cause angelegt. Standardwert ist false.

CustomCauses **customCauses()**	
Eingabe:	*Effekt/Ausgabe:*
Suchtreffer	Liefert alle Causes vom Typ "custom" der Eigenschaft.

6.15 Transaktionen

AbortTransaction **-**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Bricht die aktive Transaktion ab. Siehe Kapitel Transaktionssteuerung.

Transaction **-**	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Führt die enthaltenen Anweisungen innerhalb einer Transaktion aus. Siehe Kapitel Transaktionssteuerung.

TransactionContext transactionContext(named)	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Liest vom Transaktionskontext die Variable mit Namen @named aus, wenn eine Transaktion aktiv ist, ansonsten null
Parameter:	*Beschreibung:*
@named	KPath-Ausdruck zur Bestimmung des Variablennamens

TransactionManagerContext transactionManagerContext(named)	
Alternativnamen:	tmContext(named)
Eingabe:	*Effekt/Ausgabe:*
beliebig	Liest vom Kontext des Transaktionsmanagers die Variable mit Namen @named aus
Parameter:	*Beschreibung:*
@named	KPath-Ausdruck zur Bestimmung des Variablennamens

ReadTransaction -	
Eingabe:	*Effekt/Ausgabe:*
-	Alle folgenden Anweisungen werden innerhalb einer nur lesenden Transaktion ausgeführt

6.16 Ausgabe

Output -	
Eingabe:	*Effekt/Ausgabe:*
	Gibt Text auf einem Stream aus. Text innerhalb des Output-Elementes wird immer ausgegeben. Es können auch beliebige Anweisungen innerhalb des Textes stehen. Mit speziellen Ausgabeelementen können Daten ausgegeben werden.
Parameter:	*Beschreibung:*
[@encoding]	Encoding für den ausgegebenen Text (siehe Kapitel Encoding).

Beispiel:

```
<Output><name/> kennt </Output>
<Path path="~kennt/target()/@Name/value(ger)">
   <value/>
</Path>
```

RawOutput -	
Eingabe:	*Effekt/Ausgabe:*
	Gibt Text auf einem Stream aus. Dieses Tag kann im XML-Modus verwendet werden, wenn vorformatierte Werte ohne Escaping ausgegeben werden sollen (z.B. wenn Attributwerte bereits in XML/XHTML vorliegen). Die Ausgabe wird nicht auf XML/XHTML-Konformität überprüft.
Parameter:	*Beschreibung:*
[@encoding]	Encoding für den ausgegebenen Text (siehe Kapitel Encoding).

```
<Script output="xml">
    <MyElement>
        <RawOutput><path path="./@preformatted/value()"/></RawOutput>
    </MyElement>
</Script>
```

Hinweis: Alle folgenden Ausgabeelemente werden mit kleinem ersten Buchstaben geschrieben im Gegensatz zu Tags, die Werte zurück liefern. Daher kann es z.B. vorkommen, dass es Ausgabeelemente gibt, die genau die gleiche Logik wie gleichnamige Tags mit führendem Großbuchstaben haben, ihr Ergebnis aber nur als String auf die Ausgabe schreiben (z.B. internalName) statt diesen zurück zu liefern (InternalName). Nicht zu verwechseln mit KPath Funktionen, die auch immer mit kleinem Anfangsbuchstaben beginnen (internalName()).

Für die Ausgabetags gibt es keine äquivalenten KPath Funktionen, daher fällt die zweite Kopfzeile der Beschreibungstabellen weg.

color	
Eingabe:	*Effekt/Ausgabe:*
Topic	Gibt die im Netz definierte Farbe für dieses Topic als Web-Colour aus (z.B. #C01069)

contrastColor	
Eingabe:	*Effekt/Ausgabe:*
Topic	Gibt eine zu dem Topic passende Textfarbe aus, z.B. wenn man den Namen eines Topics auf den Hintergrund der color ausgeben will. Siehe auch color.

cr	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt einen Zeilenumbruch (ASCII-Zeichen 13 (hexadezimal 0D)) aus.

encoding	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt das aktuell gesetzte Encoding als String aus

id	
Eingabe:	*Effekt/Ausgabe:*
Objekt	gibt die ID des Objektes in der Form IDxxx_yyy aus.

idNumber	
Eingabe:	*Effekt/Ausgabe:*
Topic	Gibt die eindeutige ID des Topic als Nummer aus

indent	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt einen indent-String wiederholt aus, entsprechend der aktuellen Schachtelungstiefe der Verarbeitung
Parameter:	*Beschreibung:*
@indent	String, der zur Einrückung je Stufe ausgegeben wird

internalName	
Eingabe:	*Effekt/Ausgabe:*
Objekt	Gibt den internen Names des Objekts aus.

lf	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt einen Zeilenumbruch (ASCII-Zeichen 10 (hexadezimal 0A)) aus.

multipleOccurrences	
Eingabe:	*Effekt/Ausgabe:*
Eigenschafts-Proto	ja/nein (übersetzt) für die Angabe, ob die Eigenschaft mehrfach am Objekt vorkommen darf

name	
Eingabe:	*Effekt/Ausgabe:*
Objekt	gibt den Inhalt des Namensattributs des Objekts (in der aktuell aktiven Sprache) aus.

print	
Alternativnamen:	outputPath, path
Eingabe:	*Effekt/Ausgabe:*
beliebiges Einzelobjekt oder -wert	Schreibt den Ausgabestring des ausgewerteten Pfades in die Ausgabe
Parameter:	*Beschreibung:*
[@path]	optional statt der Eingabe auszuwertender KPath Ausdruck

separator	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt die gewünschte Separator-Zeichenkette aus
Parameter:	*Beschreibung:*
@string	Separator-Zeichenkette, obligatorische Angabe

serverTimestamp	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt die aktuelle Serverzeit als Zeitstempel aus

space	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt ein Leerzeichen aus.

time	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt die aktuelle Zeit für die lokale Zeitzone aus

timestamp	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt die aktuelle Zeit als Zeitstempel aus

timestamp_gmt	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt die aktuelle Zeit als Zeitstempel für die Zeitzone GMT aus

type	
Eingabe:	*Effekt/Ausgabe:*
beliebig	Gibt Angaben zu Typ der Eingabe/-elemente zurück. Für mengenwertige Eingaben gilt: Haben alle Elemente denselben Typ, so wird dieser nur einmal zurückgegeben, unterscheiden sie sich, so wird ein Feld mit den entsprechenden Typangaben gebildet

userName	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt den Namen des aktuellen Benutzers aus

uuid	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt die nächste UUID aus

value	
Eingabe:	*Effekt/Ausgabe:*
Attribut	Gibt den Wert des Attributs als Zeichenkette aus.

variable	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt den Wert der Variablen mit dem angegebenen Namen als Zeichenkette aus.
Parameter:	*Beschreibung:*
@name	Name der Variablen

volume	
Eingabe:	*Effekt/Ausgabe:*
-	Gibt den Namen des Wissensnetzes aus

6.17 JSON

Um eine in JSON formulierte Ausgabe zu generieren, muss im Script-Tag des Skripts der Parameter "output" auf "json" gesetzt werden

Beispiel:

```
<Script output="json">
```

Danach stehen die im folgenden beschriebenen Elemente zur Ausgabe zur Verfügung. Der Ersteller des Skripts ist für den korrekten Aufbau selbst verantwortlich, d.h. es wird nicht geprüft, ob das Skript eine valide Schachtelung der JSON-Elemente erzeugt.

Um JSON einzulesen ruft man die Funktion parseJson() bei einer Zeichenkette auf. Auf Eigenschaften eines Objekts kann man dann mit atKey(propertyName) zugreifen, auf Elemente eines Arrays mit atIndex(arrayIndex). Bei atIndex() muss man beachten, dass der Index mit 1 anfängt.

```
<SetVariable variable="name" value="var(jsonString)/parseJson()/atKey('name')"/>
```

JSONObject -	
Eingabe:	*Effekt/Ausgabe:*
	Startet ein JSON-Objekt in der Ausgabe. Hiermit wird die Verschachtelung in der JSON-Ausgabe gesteuert. Eigenschaften werden mit dem Element JSONProperty hinzugefügt.

Beispiel:

```
<Script output="json">
    <Path path="\\Wiesbaden"
        <JSONObject>
            <JSONProperty name="name" value="name()"/>
        </JSONObject>
    </Path>
</Script>
```

erzeugt folgende JSON-Ausgabe:

```
{"name": ["Wiesbaden"]}
```

JSONProperty -	
Eingabe:	*Effekt/Ausgabe:*
	Fügt dem umgebenden JSON-Objekt eine Eigenschaft mit Namen "name" und Wert "value" hinzu.
Parameter:	*Beschreibung:*
@name	Name der Eigenschaft, muss vorhanden sein
[@value]	Wert der Eigenschaft

JSONValue -	
Eingabe:	*Effekt/Ausgabe:*
	Fügt dem umgebenden JSON-Property-Objekt einen Wert hinzu, der Wert ergibt sich aus der Auswertung des KPath-Ausdrucks in "value".
Parameter:	*Beschreibung:*
@value	Wert der Eigenschaft

Beispiel:

```
<Script output="json">
   <Path path="\\Wiesbaden"
        <JSONObject>
            <JSONProperty name="name">
                <JSONValue value="name()"/>
            </JSONProperty>
        </JSONObject>
    </Path>
</Script>
```

erzeugt ebenfalls

```
{"name": ["Wiesbaden"]}
```

JSONArray -	
Eingabe:	*Effekt/Ausgabe:*
	Startet ein Array

Beispiel:

```
<Script output="json">
   <Path path="\\Wiesbaden">
        <JSONObject>
            <JSONProperty name="name">
                <JSONValue value="name()"/>
            </JSONProperty>
            <JSONProperty name="relations">
                <JSONArray>
                <Path path="./relations()/target()">
                    <Do>
                        <JSONObject>
                            <JSONProperty name="targetName" value="name()"/>
                        </JSONObject>
                    </Do>
                </Path>
                </JSONArray>
            </JSONProperty>
        </JSONObject>
    </Path>
</Script>
```

erzeugt folgende Ausgabe:

```
{
    "name": ["Wiesbaden"],
    "relations": [{
        "targetName": "Deutschland"
    },
    {
        "targetName": "Stadt"
    }]
}
```

JSONRenderTopic **jsonRenderTopic()**	
Eingabe:	*Effekt/Ausgabe:*

	Gibt eine umfassende Darstellung des Topics mit seinen Eigenschaften in JSON-formatiert zurück
Parameter:	*Beschreibung:*
[@configName]	Name der Gruppenkonfiguration, die für die Zusammenstellung und Ordnung der auszugebenden Eigenschaften verwendet werden soll. Optional, bei Weglassen wird die Standardgruppe verwendet.
[@editorConfig]	Name der Editorkonfiguration. Die Angabe ist nur notwendig, wenn es mehr als eine Editorkonfiguration gibt.

Beispiel:

```
<Script output="json">
    <Path path="\\Wiesbaden/jsonRenderTopic()"/>
</Script>
```

alternativ:

```
<Script output="json">
    <Path path="\\Wiesbaden">
            <JSONRenderTopic/>
        </Path>
</Script>
```

erzeugt:

```
{
    "name": "Wiesbaden",
    "id": "ID14152_320243140",
    "concept": "Stadt",
    "properties": [{
        "propertyType": {
            "name": "ausblenden",
            "type": "boolean"
        },
        "propertyValues": [{
            "value": "Nein",
            "id": "ID14152_197409442"
        }]
    },
    {
        "propertyType": {
            "name": "beherbergt Sehensw\u00FCrdigkeit",
            "type": "relation"
        },
        "propertyValues": [{
            "id": "ID14152_227640741",
            "targetLabel": "Kurhaus Wiesbaden",
```

```
            "targetID": "ID14893_26253985"
        },
        {
            "id": "ID14152_279432353",
            "targetLabel": "Museum Wiesbaden",
            "targetID": "ID14888_129563379"
        }]
    }]
}
```

JSONRenderList jsonRenderList()	
Eingabe:	*Effekt/Ausgabe:*
	Gibt eine tabellarische Ausgabe der Menge an Topics, die die Eingabe darstellen, in JSON-formatiert zurück
Parameter:	*Beschreibung:*
[@configName]	Name der alternativen Objektlistenkonfiguration, die für die Zusammenstellung und Ordnung der auszugebenden Eigenschaften je Topic-Zeile verwendet werden soll. Optional, bei Weglassen wird die Standard-Objektlistenkonfiguration benutzt, falls diese nicht existiert, wird eine Standard-Objektlistenkonfiguration lokal erzeugt und verwendet.
[@editorConfig]	Name der Editorkonfiguration. Die Angabe ist nur notwendig, wenn es mehr als eine Editorkonfiguration gibt.
[@concept]	Begriff der Objekte der Liste. Wenn nicht angegeben wird der Begriff aus der Liste ermittelt.
[@renderAsObjects]	Boolean, der angibt, ob eine Zeile als JSON-Objekt (true) oder als Array (false) dargestellt werden soll. Bei Objekten wird als Name der Properties der Spaltenname verwendet, als Wert der Inhalt der jeweiligen Spalte. Bei Arrays entsprechen die Werte dem Inhalt der Spalte an der jeweiligen Position. Standardwert ist false.

Beispiel: Ausgabe aller Städte, die mit dem Buchstaben "W" beginnen, gemäß der Defaultkonfiguration für Orte

```
<Script output="json">
    <Path path="//Stadt\W*/jsonRenderList()"/>
</Script>
```

alternativ:

```
<Script output="json">
    <Path path="//Stadt\W*>
            <JSONRenderList/>
        </Path>
</Script>
```

erzeugt:

```
{
    "rows": [{
        "topicID": "ID16454_381697443",
        "row": ["Wacken",
        "Schleswig-Holstein",
        "W.O.A Wacken Open Air 2011",
        ""]
    },
    {
        "topicID": "ID14130_217561445",
        "row": ["Weimar",
        "Deutschland",
        "",
        "Bauhaus, Kulturhauptstadt Europas, UNESCO-Weltkulturerbe"]
    },
    {
        "topicID": "ID14864_215244706",
        "row": ["Wetzlar",
        "Deutschland",
        "",
        ""]
    },
    {
        "topicID": "ID13598_468377392",
        "row": ["Wien",
        "\u00D6sterreich",
        "Gold Schau, Klimt-Jubil\u00E4umjahr 2012",
        ""]
    }],
    "columns": ["Name",
    "Ort",
    "Reisen",
    "Themen"]
}
```

6.18 Import und Export

ImportExternalData -	
Eingabe	*Effekt/Ausgabe*
beliebig	Importiert Daten mit Hilfe einer im Netz gespeicherten Abbildung
Parameter	*Beschreibung*
@folder	KPath des Abbildungsordners
@mapping	Name der Abbildung
[@filename]	Dateiname
[@encoding]	Zeichenkodierung
[@dbUser-name]	Benutzername
[@dbPassword]	Passwort
[@dbEnviron-ment]	Datenbank
[@dbHost-name]	Hostname der Datenbank (nur für MySQL relevant)
[@triggers]	Trigger während des Imports ausschalten [triggers="false"]
[@import-String]	Anstelle einer Datei als Datenquelle kann der zu importierende Inhalt auch in Form einer Zeichenkette übergeben werden. Anmerkung: HTML- bzw. XML-eigene Zeichen müssen maskiert werden.
<parameter name="Name" value="KPath"/> *	Wert eines Parameters innerhalb einer SQL-Query. Parameter werden durch einen Doppelpunkt, gefolgt durch einen Namen, innerhalb der Query spezifiziert. Beispiel: `select id,name,birthday from persons where id = :PersonID`

ExportTableMapping -	
Eingabe	*Effekt/Ausgabe*
beliebig	Exportiert Daten mit Hilfe einer im Netz gespeicherten Abbildung
Parameter	*Beschreibung*
@folder	KPath des Abbildungsordners

@mapping	Name der Abbildung
@filename	Dateiname
[@encoding]	Zeichenkodierung
[@dbUser-name]	Benutzername
[@dbPassword]	Passwort
[@dbEnviron-ment]	Datenbank
[@dbHost-name]	Hostname der Datenbank (nur für MySQL relevant)

ExportTopicsByMapping -	
Eingabe	*Effekt/Ausgabe*
beliebig	Exportiert Objekte, die im Elternelement ermittelt wurden, mit Hilfe einer im Netz gespeicherten Abbildung
Parameter	*Beschreibung*
@folder	KPath des Abbildungsordners
@mapping	Name der Abbildung
@filename	Dateiname
[@encoding]	Zeichenkodierung
[@dbUser-name]	Benutzername
[@dbPassword]	Passwort
[@dbEnviron-ment]	Datenbank
[@dbHost-name]	Hostname der Datenbank (nur für MySQL relevant)

```
<Script>
   <Path path="//Person\*">
      <ExportTableMapping folder="folder($MappingFolder$)" mapping="csv"/>
   </Path>
```

```
</Script>
```

Im Falle von Tabellen, die in SQL-Datenbanken schreiben, kann das obige Tag Rückgabewerte zur Verfügung stellen, die sich in eine Ausgabe lenken lassen:

```
<Script>
    <Path path="//Person\*">
        <ExportTopicsByMapping folder="folder($SQLExport$)" mapping="Personen">
            <AtKey key="updated">
                <Each>
                    <Output>updated: <path path="."></path><cr/></Output>
                </Each>
            </AtKey>
            <AtKey key="inserted">
                <Each>
                    <Output>inserted: <path path="."></path></Output>
                </Each>
            </AtKey>
        </ExportTopicsByMapping>
    </Path>
</Script>
```

6.19 Objekte sperren

- **isLockedByAnyUser()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Liefert "true" zurück, wenn das Topic durch irgendeinen Benutzer gesperrt wurde.

- **isLockedByLocalUser()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Liefert "true" zurück, wenn das Topic durch den aktuellen Benutzer gesperrt wurde.

- **isLockedByOtherUser()**	
Eingabe:	*Effekt/Ausgabe:*
Topic	Liefert "true" zurück, wenn das Topic durch irgendeinen Benutzer außer dem aktuellen Benutzer gesperrt wurde.

- **transactionContext(kemLockedTopics)**	
Eingabe:	*Effekt/Ausgabe:*
-	Enthält alle gelockten Objekte, die in der aktuellen Transaktion gelockt wurden.

6.20 Kürzester Pfad

Mit dieser Funktion werden zu jedem Ziel alle kürzesten Pfade von mindestens einem beliebigen Start-Topic aus gesucht. Dabei werden nur Relationen vom Typ der angegebenen Relationsbegriffe und nur Knoten der angegeben domains traversiert. Die Pfade traversieren max. maxLength viele Relationen hintereinander. Ist kein passender Pfad vorhanden, enthält das Ergebnis keinen Pfad zu dem Ziel-Topic.

Das Ergebnis ist eine durchnummerierte Menge geordneten Knotenmengen, die jeweils einen Pfad von einem Start- zu einem Ziel-Topic repräsentieren.

ShortestPath shortestPath**()**	
Eingabe:	*Effekt/Ergebnis:*
topics	Die Objekte, von denen aus die Pfade zu den Zielen starten. Ergebnis ist eine Abbildung (Nr->Liste) der kürzesten Pfade
Parameter:	*Effekt:*
@targets	Die Objekte, zu denen die Pfade ermittelt werden.
@maxLength	Maximale Länge eines Pfades.
[@relations]	Relationsbegriffe, die auf dem Weg zum Ziel traversiert werden dürfen. [Default: Benutzerrelation]
[@domains]	Objekttypen (protos), die auf dem Weg traversiert werden dürfen. [Default: Individuen von Wurzelbegriff für Normale Begriffe]

6.21 Logausgabe

LogTimeToRun -	
Eingabe:	*Effekt/Ausgabe:*
	Misst die Ausführungszeit der enthaltenen Anweisungen und gibt diese im Log aus
Parameter:	*Beschreibung:*
[@channel]	Name des Logchannels
[@level]	Loglevel
[@message]	KPath/Zeichenkette die Ausgegeben werden soll

```
<LogTimeToRun message="'Suchdauer'" level="10" channel="MY_DEBUG_CHANNEL">
  <SimpleSearch searchName="test" searchString="test">
    ...
  </SimpleSearch>
</LogTimeToRun>
```

ConsoleLog -	
Eingabe:	*Effekt/Ausgabe:*
	Schreibt in die Logausgabe/-datei
Parameter:	*Beschreibung:*
[@channel]	Name des Logchannels
[@level]	Loglevel
[@value]	KPath/Zeichenkette die Ausgegeben werden soll

7 KPath-Ausdrücke

Zusätzlich zu den im Kapitel KScript angegebenen KPath-Funktionen können noch folgende KPath-spezifische Sprachkonstrukte verwendet werden:

Element	Definiert für	Beschreibung
.		Aktueller Knoten
..		Übergeordneter Knoten
@name	Topic	Attribute mit dem Namen "name". Vererbung wird ab K-Infinity 3.0 berücksichtigt, d.h. es kann auch der Name eines abstrakten Attributs angegeben werden. Falls das Topic ein Prototyp ist, werden nur Attribute berücksichtigt, die direkt bei diesem Prototyp definiert sind.
@@name	Topic	(ab K-Infinity 3.0) Mögliches Attribut mit Namen "name". Falls das Topic ein Prototyp ist, werden auch Attribute berücksichtigt, die im Schema von Oberbegriffen definiert sind.
∼name	Topic	Relationen mit dem Namen "name". Vererbung wird berücksichtigt, d.h. es kann auch der Name einer abstrakten Relation angegeben werden. Falls das Topic ein Prototyp ist, werden nur Relationen berücksichtigt, die direkt bei diesem Prototyp definiert sind. Es werden keine Abkürzungsrelationen berücksichtigt.
∼∼name	Topic	(ab K-Infinity 3.0) Mögliche Relationen mit Namen "name". Falls der Topic ein Prototyp ist, werden auch Relationen berücksichtigt, die im Schema von Oberbegriffen definiert sind. Es werden auch Abkürzungsrelationen berücksichtigt.
/name	Begriff	Direkter Unterbegriff mit dem Namen "name"
//name	Begriff	Alle Unterbegriffe mit dem Namen "name"
\name	Begriff	Direkte Instanzen mit dem Namen "name". Es entfällt das trennende "/". Beispiel: //Buch\Faust
\\name	Begriff	Alle Instanzen mit dem Namen "name" des Begriffs oder eines Unterbegriffs. Es entfällt das trennende "/".

In einigen KScript-Elementen konnten in Vorversionen von KScript als Parameter nur Zeichenketten und keine KPath-Ausdrücke verwendet werden (z.B. im Parameter <name> von

CreateInstance). Um in diesem Fall die Interpretation eines Aufrufparameters als KPath-Ausdruck zu erzwingen, muss dieser in ${...} eingeschlossen werden.

Beispiel:

```
<CreateInstance concept="var(concept)" name="var(name)"/>
```

erzeugt ein Individuum des Begriffes, der in der Variablen 'concept' abgelegt ist, und benennt dieses Individuum mit 'var(name)'.

```
<CreateInstance concept="var(concept)" name="${var(name)}"/>
```

erzeugt ein Individuum des Begriffes, der in der Variablen 'concept' abgelegt ist, und benennt dieses Individuum mit dem Wert, der in der Variablen 'name' abgelegt ist.

Diese syntaktische Unzulänglichkeit war leider nicht zu vermeiden, weil ansonsten die Kompatibilität mit älteren Skripten nicht sicher gesetllt werden konnte.

Dieser Punkt betrifft alle Stellen, die im KScript-Schema mit

```
<xsd:annotation>
    <xsd:documentation>String or ${KPath}</xsd:documentation>
</xsd:annotation>
```

markiert sind.

8 KScript Schema

Das XML-Schema von KScript kann im Knowledge-Builder über das Menue **'Werkzeuge** > **Skripte** > **XML-Schema von KScript exportieren'** in eine Datei exportiert werden.

Dieser Export spiegelt den aktuellen Implementierungsstand des verwendeten Knowledge-Builders wieder und enthält insbesondere auch Skript-Funktionen, die als kundenspezifische Anpassungen nicht im Standard von K-Infinity enthalten sind und auf die deshalb in diesem Dokument auch nicht eingegangen wird.

Außerdem kann man dieses exportierte Schema verwenden, um Editoren zu konfigurieren, die Hervorhebung und automatische Ergänzung von Syntax erlauben, wie das z.B. in Eclipse integrierte Editoren machen.