EasternGraphics
visualize your business

OAP
OFML Aided Planning

Version 1.5

1. überarbeitete Fassung

Thomas Gerth, EasternGraphics GmbH (Editor)

17. Oktober 2023

Rechtliche Hinweise

Inhaltsverzeichnis

Literatur

[article] Die OFML–Schnittstellen Article und CompositeArticle (Spezifikation). EasternGraphics GmbH

[dsr] Datenstruktur und Registrierung (DSR) Spezifikation. EasternGraphics GmbH

[mt] OFML-Metatypen (MT) Spezifikation. EasternGraphics GmbH

[ofml] OFML – Standardisiertes Datenbeschreibungsformat der Büromöbelindustrie. Version 2.0, 3. überarbeitete Auflage. Industrieverband Büro und Arbeitswelt e. V. (IBA)

[property] Die OFML–Schnittstelle Property (Spezifikation). EasternGraphics GmbH

Die Spezifikationen sind über das pCon Download Center in der Kategorie *OFML Specifications* verfügbar: `https://download-center.pcon-solutions.com`

1 Einleitung

OFML Aided Planning beschreibt Konzepte, Techniken und entsprechende Datentabellen, welche eine weitgehend einheitliche Datenanlage und Umsetzung von Planungslogiken und -techniken (Inter–Produkt–Regeln) sowohl in Online- als auch Offline–Applikationen ermöglichen soll.

OAP ist als zusätzliche Schicht über den herkömmlichen OFML–Daten zu verstehen [ofml]. U.a. kann auf die Properties von Artikelinstanzen Bezug genommen werden. Damit kann die OAP–Datenanlage z.B. mit der Metadatenanlage verknüpft werden [mt].

Bestimmte Konzepte, die bislang zur Umsetzung von Planungslogiken in Offline–Applikationen verwendet wurden, z.B. OLAYER–basierte Snapping-Verfahren, werden zukünftig durch OAP ersetzt.

Hinweis:
In dieser Spezifikation beschriebene Funktionalität, die in den aktuellen Applikationen von EasternGraphics noch nicht unterstützt wird, ist grau unterlegt .

2 Konzeptionelle Grundlagen

2.1 Techniken

OAP integriert folgende Techniken:

- **Interaktoren**

  Interaktoren sind zweidimensionale grafische Symbole, die von einer Applikation über einem Objekt[1] gezeichnet werden, und bei deren Selektion durch den Anwender[2] bestimmte → *Aktionen* mit Teilen der Planung durchgeführt oder Informationen abgerufen werden können.

  Interaktoren sind objekt-spezifisch. Bei Selektion eines Objektes werden die für das Objekt definierten und im aktuellen Planungskontext *gültigen* Interaktoren angezeigt.

  Die Größe der Interaktor-Symbole ändert sich nicht in Abhängigkeit von der Entfernung der Kamera zum Objekt.

  In den OAP–Daten kann eingestellt werden, ob ein Interaktor durch andere Objekte verdeckt werden darf oder nicht. Des Weiteren können für die Interaktor-Symbole auch *Sichtbarkeitsbereiche* definiert werden, so dass sie nur aus bestimmten (sinnvollen) Blickwinkeln sichtbar sind.

  Es wird zwischen 2D- und 3D–Interaktor-Symbolen unterschieden:

  - *2D–Symbole* liegen unabhängig von der Kamera–Perspektive immer parallel zur Bildebene, unterliegen also keiner perspektivischen Verzerrung.
  - *3D–Symbole* besitzen eine definierte Ausrichtung im Raum, abhängig von der Kamera–Perspektive ergibt sich damit eine perspektivische Verzerrung.

  3D–Symbole sollen/können dort verwendet werden, wo bei Verwendung eines 2D–Symbols die Bedeutung nicht bei jeder Kamera–Perspektive zweifelsfrei erkennbar wäre, z.B. Pfeile zur Veranschaulichung einer Verschiebungsrichtung.

  Anmerkung:
  (Application) Interactors wurden vor OAP bereits im pCon.planner 8 unterstützt, wobei die Interaktoren direkt in den OFML–Daten definiert werden (mußten), s. Application Note AN-2013-001. Im Rahmen von OAP wird das in der genannten Application Note beschriebene Interaktor-Konzept jedoch angepasst und deutlich erweitert[3].

[1] grafische Darstellung eines Produktes, mehr zum Begriff *Objekt* siehe nächsten Abschn.
[2] Die Selektion erfolgt durch einen Klick auf das Interaktor-Symbol und wird auch als *Aktivierung* des Interaktors bezeichnet.
[3] Perspektivisch werden die in der Application Note beschriebenen Interaktoren damit obsolet.

- **Aktionen**

  Für bestimmte Ereignisse, z.B. die Aktivierung eines → *Application Interactors* durch den Anwender, können Aktionen definiert werden. Eine Aktion spezifiziert die Funktionalität, die beim Eintreten des Ereignisses ausgeführt werden soll.

  Im Zusammenspiel mit → *Anfügebereichen* lassen sich damit auch Inter-Produkt-Regeln abbilden, die aktuell mittels Metadatenanlage realisiert werden[4].

- **Intelligente Anfügebereiche**

  Intelligente Anfügebereiche erweitern das Konzept der herkömmlichen OFML–Anfügepunkte: Außer Punkten können auch Linien und Flächen, ggf. mit Raster, angegeben werden. Zudem kann der Bereich mit *Aktionen* (siehe oben) verknüpft werden, die auszuführen sind, wenn der Anfügebereich zur Verbindung zweier Objekte verwendet wurde bzw. wenn diese Verbindung wieder gelöst wird.
  Mit den Intelligenten Anfügebereichen können/sollen zukünftig Snapping-Verfahren unterstützt werden, wie sie aktuell im pCon.planner auf der Basis der OLAYER `D2SNAP` sowie `ATTACH` & `ORIGIN` realisiert sind.

2.2 Begriffe

Folgende Begriffe von fundamentaler Bedeutung werden in diesem Dokument verwendet:

- **Artikel vs. Artikelvariante**

  Ein *Artikel* (Synonym: *Produkt*) ist ein durch einen Hersteller oder Anbieter erzeugbarer bzw. erzeugter und/oder zum Verkauf angebotener Gebrauchsgegenstand.

  Kann ein Teil der Eigenschaften eines Artikels durch den Käufer bzw. Anwender der OFML–Applikation festgelegt werden, spricht man von einem *konfigurierbaren Artikel*. Die konkrete Ausprägung eines Artikels in Bezug auf seine konfigurierbaren Eigenschaften wird dann als *Artikelvariante* bezeichnet (Synonym: *Artikelkonfiguration*).

  Innerhalb eines Herstellers/Anbieters wird ein Artikel durch einen alphanumerischen Code, die *Artikelnummer*, eindeutig identifiziert. Die Ausprägungen der konfigurierbaren Eigenschaften werden im *Variantencode* codiert, wobei verschiedene Schemata zur Anwendung kommen.

  Da die meisten Artikel konfigurierbar sind, wird in diesem Dokument der Einfachheit halber in der Regel nur der Begriff *Artikel* verwendet, auch wenn im Falle eines konfigurierbaren Artikels eigentlich die *Artikelvariante* gemeint ist. Nur an den Stellen, wo eine Unterscheidung zwingend ist, wird explizit *Artikelvariante* verwendet.

- **Artikelrepräsentation vs. OFML–Instanz**

  Als *Artikelrepräsentation* (Synonyme: *Planungselement*, *Objekt*) wird das Objekt bezeichnet, welches einen → *Artikel* in einem Planungs- bzw. Konfigurationssystem grafisch repräsentiert.

  Die Konfigurierung eines (konfigurierbaren) → *Artikels* erfolgt auf der Basis von OFML–*Properties*. Dazu muss eine OFML–*Artikelinstanz* (kurz OFML–*Instanz*) erzeugt werden. OFML–Instanzen werden auch zur Ermittlung von diversen Artikelinformationen (Texte, Preise etc.) benötigt.

  Eine OFML–Instanz beinhaltet auch eine grafische Darstellung des Artikels. Aus technischen Gründen werden OFML–Instanzen in der Regel aber nicht als → *Artikelrepräsentation* im Planungssystem verwendet. In diesem Fall muß zur Konfigurierung eines Artikels bzw. zur Ermittlung von Artikelinformationen temporär eine (für den Anwender nicht sichtbare) OFML–Instanz erzeugt werden.

  In den Online–Applikationen (auf der Basis des EAIWS) besitzt ein Artikel zwei Repräsentationen: die grafische im Client (Planungselement) und eine kaufmännische im Warenkorb, der vom Server verwaltet wird. Die Repräsentation im Warenkorb des Servers wird deswegen in Abgrenzung zum Planungselement als *Basket–Instanz* bezeichnet. Dem Server obliegt auch die Erzeugung einer

[4] Artikelpolymorphie und Intra-Produkt-Regeln können und sollen auch weiterhin mittels Metadaten abgebildet werden.

(temporären) OFML–Instanz zu einer Basket–Instanz, wenn dies zur Erfüllung einer Anforderung seitens des Clienten notwendig ist.

- **aktives vs. passives Planungselement**

  Beim Einfügen eines neuen Elementes in die Planung bzw. beim Entfernen eines Planungselementes sowie beim Verschieben eines Planungselementes kann man 2 Rollen unterscheiden, die die Planungselemente spielen: Das Element, das eingefügt, gelöscht oder bewegt wird, spielt die aktive Rolle und wird entsprechend als *aktives Planungselement* bezeichnet. Die anderen Elemente spielen eine passive Rolle und werden entsprechend als *passive Planungselemente* bezeichnet.

  Diese Unterscheidung ist insbesondere relevant in Hinblick auf die Anfügepunkte/-bereiche: einige der Anfügebereiche eines Planungselementes können/sollen nur verwendet werden, wenn das Element die aktive Rolle spielt, andere nur, wenn es eine passive Rolle spielt. Die Anfügebereiche werden ensprechend auch als aktiv bzw. passiv bezeichnet, wobei es Anfügebereiche geben kann, die in beiden Rollen verwendet werden können.

2.3 Sonstige Aspekte / Definitionen

- **Nachbarschaft**

  Eine *Nachbarschaftsbeziehung* (Synonym: Verbindung) zwischen zwei Planungselementen liegt vor, wenn es (mindestens) ein Paar von Anfügebereichen der beiden Elemente gibt, die logisch und geometrisch zueinander passen.

- **Variantenabhängigkeit**

  Abhängigkeiten in den OAP–Daten von einer Artikelvariante werden auf der Basis von OFML–Properties abgebildet[5].

  Dazu wird der sogenannte *Property–Variantencode*, abgekürzt *PropVarCode*, eingeführt, der die aktuellen Werte der OFML–Properties einer Artikelvariante codiert (Details siehe Feldtyp *PVC* im Abschn. 3.2):

  ```
  <Property>=<Value>;<Property>=<Value>;...
  ```

  In den OAP–Tabellen (s. Abschn. 4) sind Felder vorgesehen, in denen entweder ein (teilbestimmter) PropVarCode oder ein Ausdruck angegeben werden kann, der mit OFML–Properties operiert.

  Eine Konsequenz ist, dass zur Auswertung der OAP–Daten für eine gegebene Artikelvariante deren PropVarCode bekannt sein muß, welcher von der OFML–Instanz des Artikels erfragt werden muß[6]. Es obliegt der Verantwortung der Applikationen, durch Verwendung geeigneter Caching- und Persistenz–Techniken die Anzahl der notwendigen OFML–Instanzerzeugungen zu minimieren und damit für eine gute Performanz zu sorgen[7].

- **OAP–Typen**

  Ein OAP–Typ umfasst die Menge aller Artikel bzw. Artikelvarianten, die im Rahmen von OAP dieselben Eigenschaften besitzen und gleich behandelt werden sollen. OAP–Typen werden in der Tabelle `Type` definiert (s. Abschn. 4.1.1).

  In entsprechenden Mapping–Tabellen werden spezifische Artikel bzw. Artikelvarianten oder auch Meta-Typen einem konkreten OAP–Typ zugeordnet.

[5] OAP–Daten können somit z.B. auch auf eine Metatyp-basierte Datenanlage bzw. auf speziell programmierte OFML–Daten aufgesetzt werden.

[6] Dies erfolgt mittels der neuen Methode *getPropVarCode(pState(Int))* der Basisklasse *OiPlElement*, wobei als Status `0` zu verwenden ist, damit auch die unsichtbaren (und zum Teil grafik-relevanten) Properties im Code repräsentiert werden.

[7] Zur Realisierung flüssiger Interaktionstechniken (z.B. Drag'n'Drop) beim Einfügen eines Artikels in eine Planung kann es u.U. notwendig werden, in den Katalogdaten zu einem Artikel bereits seinen PropVarCode anzugeben. Der Einfügevorgang (Platzierung) würde/müsste dann mit einer Ersatzgrafik stattfinden (beginnen), und die Erzeugung der OFML–Instanz findet dann erst später oder ganz am Ende des Vorgangs statt, wenn der Artikel eingefügt werden kann und die Position festgelegt ist. Dazu wären dann jedoch auch noch Erweiterungen in den Spezifikationen von XCF bzw. OAS erforderlich.

3 Allgemeine Festlegungen

3.1 Bestimmungen zum Format

Als physisches Austauschformat zwischen OFML–konformen Applikationen werden CSV–Tabellen (comma separated values) verwendet. Hierzu gelten folgende Bestimmungen:

1. Jede der unten beschriebenen Tabellen ist in genau einer Datei enthalten. Der Dateiname wird durch den Präfix „`oap_`“, den spezifizierten Tabellennamen und den Suffix „`.csv`“ gebildet, wobei der Tabellenname komplett klein geschrieben wird.
2. Als Zeichensatz wird `UTF-8` verwendet[8]. Am Anfang der Datei kann optional das Byte Order Mark angegeben werden.
3. Jede Zeile der Datei repräsentiert einen Datensatz[9].
   Leerzeilen, d.h., solche aus null oder mehr Leerzeichen (U+0020) oder Tabulator (U+0009), werden ignoriert.
   Zeilen, die mit einem Doppelkreuz ('`#`'=U+0023) beginnen, werden als Kommentar interpretiert und ebenfalls ignoriert.
4. Die Repräsentationen der einzelnen Felder eines Datensatzes werden durch Semikolon ('`;`'=U+003B) voneinander getrennt.
5. Der Wert eines Feldes besteht aus null oder mehr Unicode-Zeichen mit gültiger UTF-8–Codierung, ausgenommen die Steuer-Zeichen U+0000..U+001F sowie U+007F..U+009F.
6. Die Repräsentation eines Feldes wird aus dem Wert des Feldes gebildet, indem jedes doppelte Anführungszeichen ('`"`'=U+0022) durch zwei doppelte Anführungszeichen ersetzt und die resultierende Zeichenkette in doppelte Anführungszeichen eingeschlossen wird. Wenn der Wert nicht mit einem doppelten Anführungszeichen beginnt und kein Semikolon ('`;`'=U+003B) enthält, kann alternativ der Wert des Feldes unverändert als Repräsentation übernommen werden.

3.2 Tabellenbeschreibung und Feldtypen

Bei den Tabellenbeschreibungen wird ein Feld eines Datensatzes durch folgende Attribute spezifiziert:

- Nummer
- Bezeichner
- Kennzeichen, ob das Feld zum Primärschlüssel der Tabelle gehört[10]
- Feldtyp (s.u.)
- maximale Länge des Feldes (Anzahl der Zeichen)[11]
- Kennzeichen, ob das Feld unbedingt gefüllt sein muss (Pflichtfeld)

In Schlüsselfeldern einer Tabelle darf es keine zwei Werte geben, die sich nur in der Schreibweise unterscheiden[12].

[8] Als Normalform sollte NFC (Normalization Form Canonical Composition) verwendet werden.
[9] Eine Zeile wird entweder durch ein LF-Zeichen (U+000A) oder durch eine Folge aus CR (U+000D) und LF abgeschlossen.
[10] Zu einem Primärschlüssel darf es nur einen Datensatz in der Tabelle geben.
[11] Bei CSV–Datensätzen bestehen prinzipiell zwar keine Beschränkungen der einzelnen Feldlängen, bei bestimmten Feldern des Feldtyps `Char` werden hier jedoch sich aus dem Verwendungszweck ergebende maximal mögliche bzw. sinnvolle Längen angegeben. Darüber hinaus sind bei der Datenanlage ggf. weitergehende Beschränkungen zu beachten, die durch das im Datenanlageprozeß verwendete Programm auferlegt werden.
[12] hinsichtlich Groß- und Kleinschreibung

Folgende **Feldtypen** sind definiert:

Text Text

Es sind alle Zeichen gemäß Bestimmung 5 (s.o.) erlaubt, außer dem geschützten Leerzeichen (non-breaking space=U+00A0) und dem bedingten Trennstrich (U+00AD).

Char Zeichenkette

Es sind alle Zeichen aus dem ASCII-Zeichensatz erlaubt.

PVC Property-Variantencode

In einem Property-Variantencode werden Properties und ihre Werte in der Form

`<property_key>=<property_value>`

dargestellt, wobei einzelne Property-Darstellungen durch ein Semikolon (';'=U+003B) getrennt werden.

Property-Schlüssel werden ohne vorangestelltes '`@`'-Zeichen (U+0040) angegeben.
Werte werden gemäß den Regeln für literale OFML–Konstanten dargestellt.

Ein Property-Variantencode muß nicht zwingend alle Properties einer gegebenen OFML–Instanz umfassen, darf/kann andererseits aber auch nicht-sichtbare Properties enthalten.

Lang Sprach–Code

Die Angabe setzt sich zusammen aus dem zweistelligen Sprachkürzel nach ISO 639-1 und dem zweistelligen Länder- bzw. Regionenkürzel nach ISO 3166-1 (ALPHA-2), getrennt durch einen Bindestrich (U+002D)[13]. Die Angabe des Regionenkürzels ist optional (s. Abschn. 3.3).

Bsp.:
`en-US` für amerikanisches English
`en-GB` für britisches English

Wenn ein Datenelement für beliebige Sprachen verwendet werden kann, so ist das entsprechende Feld leer zu lassen.

Symbol Symbol

Erlaubt sind alle alphanumerischen Zeichen aus dem ASCII-Zeichensatz ('`0`'..'`9`'=U+0030..U+0039, '`A`'..'`Z`'=U+0041..U+005A, '`a`'..'`z`'=U+0061..U+007A) sowie der Unterstrich ('`_`'=U+005F), wobei das erste Zeichen keine Ziffer sein darf.

ID Identifikator

Erlaubt sind alle alphanumerischen Zeichen aus dem ASCII-Zeichensatz ('`0`'..'`9`'=U+0030..U+0039, '`A`'..'`Z`'=U+0041..U+005A, '`a`'..'`z`'=U+0061..U+007A) sowie das Minuszeichen ('`-`'=U+002D) und der Unterstrich ('`_`'=U+005F).

Identifikatoren müssen an jeder Stelle in gleicher Schreibweise[14] verwendet werden.

OID Objekt-Identifikator

Ein Objekt-Identifikator referenziert ein spezielles Objekt oder eine Menge von Objekten.

Ein Objekt-Identifikator ist entweder ein einfacher Identifikator, der dem Feldtyp `ID` entspricht, oder ein hierarchischer Name, bei dem die einzelnen Hierarchiebenen einen einfachen Identifikator darstellen und die Ebenen untereinander durch einen Punkt ('`.`'=U+002E) getrennt sind.

Hierarchische Objektnamen können verwendet werden, wenn es in der Planung Artikel gibt, die untereinander in einer Oberartikel-Unterartikel-Beziehung stehen. Die vorderen Namensegmente kennzeichnen dabei die übergeordneten Ebenen. Wenn der Identifikator einer übergeordneten Ebene eine Menge von Objekten referenziert, wird der Identifikator der untergeordneten Ebene auf alle Objekte dieser Menge angewendet (Produktmenge).

[13] Diese Festlegung orientiert sich an der Spezifikation der IETF für Language Tags. Kleinschreibung des Sprachkürzel und Großschreibung des Länderkürzels ist zu beachten!
[14] hinsichtlich Groß- und Kleinschreibung

ID_List komma-separierte Liste von Identifikatoren (Feldtyp *ID*)

OID_List komma-separierte Liste von Objekt-Identifikatoren (Feldtyp *ID*)

OFML Bezeichner entsprechend OFML Standard (Part III) [ofml]

Mögliche Angaben sind:

- OFML–Package
- OFML–Schnittstelle
- vollqualifizierter OFML–Typ (Klasse)

Int nicht-negative Ganzzahl

Erlaubt sind alle Ziffern aus dem ASCII-Zeichensatz (U+0030..U+0039).

Num numerischer Wert

Erlaubt sind alle Ziffern aus dem ASCII-Zeichensatz (U+0030..U+0039) sowie der Dezimalpunkt ('.'=U+002E)[15], optional das Minuszeichen ('-'=U+002D) an erster Stelle.

Bool boolescher Wert

'1' – true, '0' – false

NumExpr numerischer Ausdruck

Es wird erwartet, dass das Ergebnis der Auswertung des Ausdrucks ein numerischer Wert gemäß Feldtyp *Num* ist.

Mehr zu Ausdrücken siehe unten.

BoolExpr boolescher Ausdruck

Es wird erwartet, dass das Ergebnis der Auswertung des Ausdrucks ein boolescher Wert ist:

- Das Ergebnis der Auswertung des Ausdrucks wird als (eindeutig) *true* angesehen, wenn es entweder einen numerischen Typ hat und der Wert ungleich Null ist, oder wenn es vom Typ String ist und der Wert eine nicht leere Zeichenkette ist.
- Das Ergebnis der Auswertung des Ausdrucks wird als (eindeutig) *false* angesehen, wenn es entweder einen numerischen Typ hat und der Wert gleich Null ist, oder wenn es vom Typ String ist und der Wert eine leere Zeichenkette ist.
- In allen anderen Fällen ist das Ergebnis *undefiniert*.

Mehr zu Ausdrücken siehe unten.

Die lexikalische Struktur und die Syntax von OAP–*Ausdrücken* (Feldtypen *NumExpr* und *BoolExpr*) ist ausführlich im Anhang A beschrieben[16]. Diese entsprechen weitgehend den Ausdrücken, die im Part III des OFML Standards spezifiziert sind. Darüber hinaus weisen OAP–Ausdrücke u.a. folgende Besonderheiten auf:

- In OAP–Ausdrücken stehen die OFML–Properties des im jeweiligen Auswertungskontext aktiven Planungselementes (und ggf. weiterer Objekte) unter ihrem Namen als Variablen zur Verfügung.
- Die Funktion

  `methodCall(<Action-ID>)`

  kann verwendet werden, um OFML–Methoden aufzurufen. Das Argument der Funktion ist die ID einer Aktion[17] vom Typ `MethodCall`, welche den Methodenaufruf spezifiziert. (Details siehe A.4.15.)

[15] wobei der Dezimalpunkt nur einmal auftreten darf
[16] Die dort beschriebenen Datentypen sind nicht identisch mit den hier beschriebenen Feldtypen.
[17] oder ein Ausdruck, der eine ID liefert

Treten bei der Auswertung von Ausdrücken Fehler auf (z.B. Syntaxfehler oder Referenzen auf nicht existente Properties), gelten folgende Bestimmungen:

- Beim Feldtyp *NumExpr* wird der Wert `0.0` angenommen.
- Beim Feldtyp *BoolExpr* ist das Ergebnis *undefiniert* (nicht eindeutig wahr oder falsch).
  Für jedes Feld dieses Typs wird in der Spezifikation der betreffenden Tabellen das Verhalten im Fall eines undefinierten Ausdrucks explizit festgelegt.

3.3 Sprachspezifische Datenelemente

Für die Verwendung von Sprach- und Regionenkürzel im Sprach-Code im Feld `Language` (Typ *Lang*) der entsprechenden Tabellen[18] gelten folgende Bestimmungen:

- Für jeden Tabelleneintrag mit einem Sprach-Code, der Sprach- **und** Regionenkürzel enthält, sollte auch ein Tabelleneintrag mit einem Sprach-Code vorhanden sein, der nur das betreffende Sprachkürzel enthält. Dieser Eintrag dient als Fallback für Anwendungen, die den regionsspezifischen Sprach-Code nicht unterstützen.
- Welche Sprache (Region) für den Fallback-Eintrag verwendet wird, liegt im Ermessensspielraum der Hersteller/Datenanleger.
- Sind beide Tabelleeinträge inhaltlich identisch, kann (und sollte) der Eintrag mit dem regionsspezifischen Sprach-Code entfallen.

  Bsp.:
  Ist eine Text–Ressource sowohl für amerikanisches als auch für britisches English angelegt, und wird als Fallback für `en` der Text im britischen English verwendet[19], wird im Feld `Language` des Tabelleeintrags mit dem Text im amerikanischen English der Sprach-Code `en-US` angegeben, während für den Tabelleeintrag mit dem Text im britischen English als Sprach-Code nur `en` angegeben wird.

Bei der Auswahl eines Tabelleneintrags aus den Einträgen, die zu einem gegebenen Suchschlüssel passen (Text- bzw. Image–ID), geht die Applikation dann wie folgt vor:

1. Eine Anwendung, in der Sprache und Region eingestellt ist, verwendet den Eintrag mit dem passenden Sprach-Code, der Sprach- **und** Regionenkürzel enthält.
2. Ist kein passender Eintrag mit dem regionsspezifischen Sprach-Code vorhanden, oder ist in der Anwendung nur die Sprache, aber nicht die Region eingestellt, wird der Eintrag mit dem passenden Sprach-Code verwendet, der nur das betreffende Sprachkürzel enthält.
3. Ist auch kein passender Eintrag mit dem einfachen (nur aus dem Sprachkürzel bestehenden) Sprach-Code vorhanden, wird der Eintrag mit leerem Feld `Language` verwendet (insofern vorhanden)[20].

[18] aktuell `Text` und `Image`
[19] z.B. in der Annahme, daß die OFML–Daten vor allem in Europa verwendet werden
[20] Für Texte sollte immer eine Sprache angegeben werden, bei Bildern hingegen ist das Feld `Language` in der Regel leer.

3.4 Bestimmungen zum Ablageort

Die Tabellen werden standardmäßig im Vertriebsgebiet der betreffenden OFML–Serie abgelegt:

`<data>/($manufacturer)/($program)/($region)/($version)/oap`

Die Tabellen müssen in eine EBase-Datenbank namens `oap.ebase` compiliert werden.

Sollen serienübergreifende, herstellerweite Logiken abgebildet werden, so müssen die OAP–Daten in einer spezifischen Serie des Herstellers (z.B. *global*) abgelegt werden. In den Registrierungsdateien der betreffenden Produkt-Serien muß dann mittels des Schlüssels `oap_program` auf diese Serie verwiesen werden[21].

3.5 Bestimmungen zur Persistenz von Artikeln

Damit die Artikel in den Online–Applikationen verarbeitet werden können und zur performanten Verarbeitung im pCon.planner muss in den Registrierungsdaten [dsr] für den Schlüssel `persistency_form` der Wert `STATECODES` angegeben werden.

Der Zustand aller Objekte (Artikel) muss entsprechend vollständig durch die in der OFML–Schnittstelle *Article* definierten Zustandcodes beschrieben werden. Das gilt auch für ggf. verwendete Teilplanungen[22].

[21] Die Syntax der Serienangabe entspricht der Syntax für den Schlüssel `catalogs`, also `::($manufacturer)::($program)::`

[22] Insofern Teilplanungen keine realen Artikel darstellen, müssen sie also als sogenannte *Pseudo–Artikel* angelegt werden.

4 Die Tabellen

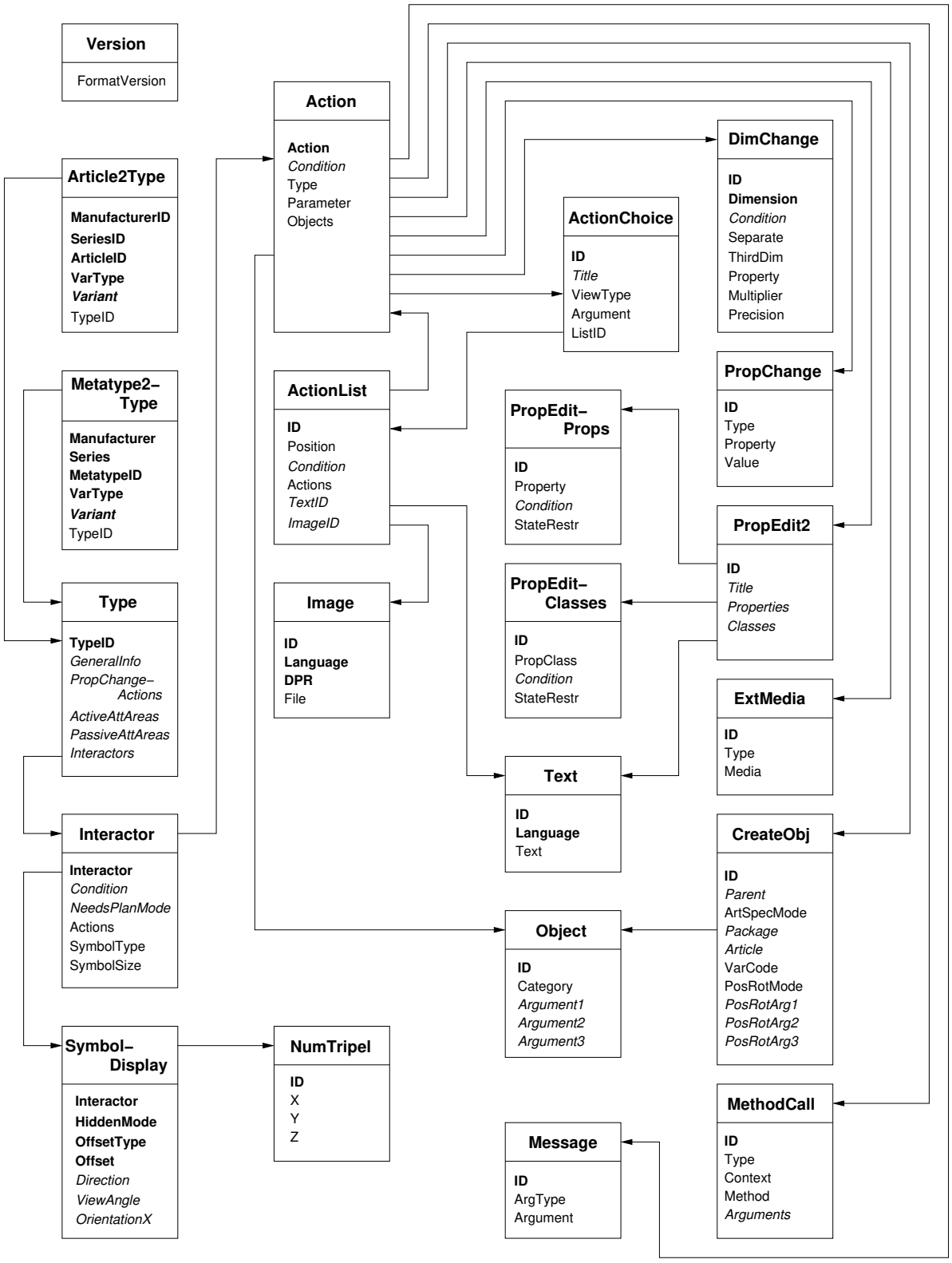


Abbildung 1: Tabellen-Übersicht

Schlüsselfelder sind durch Fettdruck hervorgehoben und Felder, die keine Pflichtfelder sind, durch Kursivdruck.

4.1 OAP–Typen

4.1.1 Die Typ–Tabelle

Tabellenname: `Type`
Pflichttabelle: ja

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	TypeID	X	Char		X	ID des OAP–Typen
2.	GeneralInfo		ID			Allgemeine Informationen
3.	PropChangeActions		ID_List			Propertyänderung-Aktionen
4.	ActiveAttAreas		ID_List			Aktive Anfügebereiche
5.	PassiveAttAreas		ID_List			Passive Anfügebereiche
6.	Interactors		ID_List			Interaktoren

Anmerkungen:

- Die ID in Feld 2 verweist in die Tabelle der allgemeinen Angaben (`GeneralInfo`).
- Die IDs im Feld 3 verweisen auf die Tabelle `Action`.

  Die *PropChangeActions* werden ausgeführt, nachdem im Eigenschaftseditor der Applikation eine Property des Artikels geändert wurde. Sie werden *nicht* ausgeführt, wenn eine Property-Änderung im Rahmen einer Aktion der Typen *PropValue* und *PropEdit2* stattfand.
  Die Aktionen werden in der Reihenfolge der IDs ausgeführt!
- Die IDs in den Feldern 4 und 5 verweisen auf die Tabelle `AttachArea`.

  Die aktiven Anfügebereiche werden verwendet, wenn das den Artikel repräsentierende Planungselement die aktive Rolle spielt, entsprechend die passiven Anfügebereiche bei einer passiven Rolle. (Ein Anfügebereich, der in beiden Rollen relevant ist, muss entsprechend in beiden Feldern referenziert werden.)
  Sind keine aktiven Anfügebereiche angegeben, kann der Artikel nur frei platziert werden, d.h. der Snapping-Mechanismus der Applikation tritt nicht in Kraft. Sind keine passiven Anfügebereiche angegeben, kann an den Artikel nicht mittels des Snapping-Mechanismus der Applikation angefügt werden.
- Die IDs im Feld 6 verweisen auf die Tabelle `Interactor`.

4.1.2 Die Mapping-Tabellen

Tabellenname: `Article2Type`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ManufacturerID	X	Char	16	X	kaufm. Herstellerkürzel
2.	SeriesID	X	Char	16	X	kaufm. Serienkürzel
3.	ArticleID	X	Char		X	Artikelnummer
4.	VarType	X	Symbol		X	Art der Variantenspezifikation
5.	Variant	X	Char			Variantenspezifikation
6.	TypeID		ID		X	ID des zugeordneten OAP–Typen

Tabellenname: `Metatype2Type`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	Manufacturer	X	Char		X	OFML–Herstellerkürzel
2.	Series	X	Char		X	OFML–Serienkürzel
3.	MetatypeID	X	Char		X	Metatyp–ID
4.	VarType	X	Symbol		X	Art der Variantenspezifikation
5.	Variant	X	Char			Variantenspezifikation
6.	TypeID		ID		X	ID des zugeordneten OAP–Typen

In beiden Tabelle kann das Feld 5 verwendet werden, um OAP–Typen spezifischen Artikelvarianten zuzuordnen. Die Art der Variantenspezifikation wird dabei im Feld 4 festgelegt. Aktuell werden folgende drei Arten unterstützt:

None Der Eintrag ist für alle Artikelvarianten gültig.

Das Feld 5 ist in diesem Fall leer (bzw. ein ggf. vorhandener Inhalt wird ignoriert).

Expr Die Variante wird durch einen booleschen Ausdruck definiert (in dem bestimmte Properties verwendet werden)[23].

PVC Die Variante wird durch einen *PropVarCode* definiert[24].

In der Regel reicht dabei die Angabe eines teilbestimmten PropVarCodes aus, der nur die Properties codiert, die zur Unterscheidung der Artikelvarianten notwendig sind.

Die Mapping-Einträge für einen gegebenen Artikel bzw. einen gegebenen Metatypen dürfen entweder nur *Expr* oder *PVC* zur Angabe von Artiklevarianten benutzen, und dürfen nur einen Eintrag mit dem Wert *None* im Feld 4 beinhalten!

Die Verfahren zur Auswahl des passenden Mapping-Eintrags sind im folgenden Abschnitt beschrieben.

4.1.3 Ermittlung des passenden Mapping-Eintrags

Zuerst wird die Menge aller Tabelleneinträge ermittelt, bei denen Hersteller, Serie und Artikelnummer bzw. Metatyp–ID übereinstimmen.

Der Tabellenzugriff schlägt fehl, wenn kein passender Tabelleneintrag gefunden wird, oder wenn Einträge in der resultierenden Menge unterschiedliche Arten der Variantenspezifikation verwenden (*Expr* und *PVC*), oder wenn es in der resultierenden Menge mehr als einen Eintrag mit dem Wert *None* in Feld 4 gibt.

Variantenspezifikation mittels booleschem Ausdruck

Aus den im ersten Schritt ermittelten Einträgen werden alle Einträge mit Angabe einer Artikelvariante entfernt, bei denen die Auswertung des Ausdrucks im Feld 5 nicht eindeutig *true* ergibt.

Der Tabellenzugriff schlägt fehl, wenn die resultierende Menge leer ist oder mehr als einen Eintrag mit Angabe einer Artikelvariante enthält.
Enthält die resultierende Menge einen Eintrag mit Angabe einer Artikelvariante und einen ohne (*None* im Feld 4), wird der Eintrag mit Artikelvariante verwendet.

Variantenspezifikation mittels PropVarCode

Die (teilbestimmten) PropVarCodes (Feld 5) der im ersten Schritt ermittelten Einträge mit Angabe einer Artikelvariante werden mit dem PropVarCode der aktuell bearbeiteten Artikelinstanz vergleichen. Dabei werden für jedes Merkmal, das in beiden Codes enthalten ist, die jeweiligen Werte miteinander vergleichen.

[23] Das Feld 5 wird dann wie ein Feld vom Typ *BoolExpr* behandelt.
[24] Das Feld 5 wird dann wie ein Feld vom Typ *PVC* behandelt.

Zwei Merkmalswerte sind gleich wenn

- beide Werte Symbolliterale sind und den gleichen symbolischen Wert haben
- beide Werte Zeichenkettenliterale sind und beide Zeichenkettenliterale die gleiche Zeichenkette repräsentieren
- beide Werte `NULL` sind
- beide Werte ein numerisches Literal sind (Integer oder Gleitkomma), einen gültigen Wert besitzen[25] und den gleichen numerischen Wert repräsentieren[26]

(Details zu den Literalen siehe Anhang A.3.2.)

Wenn nicht alle verglichenen Merkmalswerte gleich sind, dann wird der Tabelleneintrag aus der Menge der weiter zu betrachtenden Einträge entfernt[27].

Abschließend wird die Menge auf die Tabelleneinträge mit den meisten übereinstimmden Merkmalswerten reduziert[28].

Der Tabellenzugriff schlägt fehl, wenn die resultierende Menge nicht genau einen Eintrag enthält.

4.2 Die NumTripel–Tabelle

Ein Tripel dient zur Angabe der Koordinaten einer Position, von Rotationsachsen, von Verschiebevektoren und von sonstigen dreidimensionalen Parametern.
Tripel werden aus diversen Tabellen dieser Spezifikation referenziert.

Tabellenname: `NumTripel`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Tripels
2.	X		NumExpr		(X)	X-Wert
3.	Y		NumExpr		(X)	Y-Wert
4.	Z		NumExpr		(X)	Z-Wert

4.3 Allgemeine Angaben

Noch nicht unterstützt.

4.4 Anfügebereiche

Noch nicht unterstützt.

4.5 Zueinander passende Anfügebereiche

Noch nicht unterstützt.

[25] Details zu den gültigen Wertebereichen von numerischen Literalen siehe Anhang A.
Eine Konsequenz der Regelung ist, dass der Vergleich *false* ergibt, wenn beide Werte ungültig sind, auch wenn sie durch die gleiche Zeichenfolge repräsentiert werden.

[26] Die einzige Einschränkung der aktuellen Implementierung beim Vergleich dezimaler Ganzzahlen und Gleitkommazahlen besteht darin, dass der Betrag eines eventuell vorhandenen Exponenten nicht größer als 9999 sein darf (in welchem Fall der Wert als ungültig angesehen wird). Abgesehen davon wird immer der exakte dezimale Wert ohne Einschränkung der Genauigkeit verglichen.

[27] Da bei Tabelleneinträgen ohne Angabe einer Artikelvariante (*None* im Feld 4) kein Werte verglichen werden (können), werden diese Einträge also mit in die Menge der weiter zu betrachtenden Tabelleneinträge übernommen.

[28] Bei mehrfachem Auftreten des gleichen Merkmals in einem PropVarCode wird das Merkmal nur einmal gezählt.

4.6 Interaktoren

Tabellenname: `Interactor`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	Interactor	X	ID		X	Bezeichner des Interaktors
2.	Condition		BoolExpr			Gültigkeitsbedingung
3.	NeedsPlanMode		BoolExpr			<abgekündigt/obsolet>
4.	Actions		ID_List		X	Aktionen
5.	SymbolType		Symbol		X	Typ des Interaktor-Symbols
6.	SymbolSize		Symbol		X	Größe des Interaktor-Symbols

Anmerkungen zu den Feldern:

- Der Interaktor ist *gültig*, wenn das Feld 2 leer ist oder wenn die Evaluierung des in dem Feld angegebenen Ausdrucks eindeutig *true* ergibt.

- Die IDs im Feld 4 verweisen auf die Tabelle `Action`.

  Die Aktionen werden bei Betätigung des Interaktors in der Reihenfolge der IDs ausgeführt. Dabei wird eine Aktion übersprungen, wenn deren Ausführungsbedingung aktuell nicht erfüllt ist, oder wenn sie anderweitig *ungültig* ist[29]. (Der Kontext für die Auswertung der Ausführungsbedingung(en) der Aktionen wird nach jeder Ausführung einer Einzelaktion aktualisiert.)

  Ein Interaktor wird nur dann angezeigt, wenn es (aktuell) mindestens eine gültige Aktion gibt[30].

  Tritt beim Zugriff auf die Aktionsparameter in der OAP-Datenbank ein Fehler auf, oder schlägt die Ausführung einer Aktion fehl, wird die Abarbeitung der Aktionsliste abgebrochen!

  Die Abarbeitung der Aktionsliste wird ebenfalls nach einer Aktion vom Typ *SelectObj* abgebrochen (s. Abschn. 4.7).

- Das *Symbol* eines Interaktors ist ein Piktogramm, welches die (hauptsächliche) Wirkung des Interaktors veranschaulicht. Im Interesse eines einheitlichen Designs des GUI der Applikationen wird den Interaktoren im OAP nicht direkt eine Bilddatei für das Symbol zugewiesen. Vielmehr wird ein abstrakter, vordefinierter Symbol–Typ festgelegt (Feld 5). Von der Applikation wird zur Darstellung des Symbols dann ein zum Typ passendes Bild verwendet.

  Die Position sowie ggf. Ausrichtung und Sichtbarkeitsbereich des Symbols wird in der unten beschriebenen Tabelle `SymbolDisplay` festgelegt.

  Bestimmte Aktionen (z.B. *DimChange*, s. Abschn. 4.7) werden in einem speziellen Applikationsmodus ausgeführt, in welchem applikationsspezifische, nicht im OAP definierte Interaktoren zur Anwendung kommen. Während dieser Modi sind die OAP–Interaktoren nicht sichtbar.
  Symbol–Typen für Interaktoren, deren erste Aktion die Aktivierung eines speziellen Applikationsmodus bewirkt, werden in der folgenden Auflistung mit *App* gekennzeichnet. Zu einem gegebenen Zeitpunkt, d.h. unter Berücksichtigung der Gültigkeitsbedingungen der Interaktoren, sollte es immer nur je einen Interaktor mit einem dieser speziellen Symbol–Typen geben.

  Folgende Symbol–Typen sind definiert:

Add	Hinzufügen von Planungselementen
Attention	Ausgabe einer wichtigen Information (mittels einer Aktion vom Typ *Message*)
ChangeDimHorizontal	Ändern der horizontalen Ausdehnung
ChangeDim2Left	Verkleinerung der horizontalen Ausdehnung
ChangeDim2Right	Vergrößerung der horizontalen Ausdehnung

[29] z.B. nicht unterstützter Aktionstyp oder falsche bzw. fehlende Aktionsparameter
[30] In frühen Phasen der Projektentwicklung ist ggf. der Dummy-Aktionstyp *NoAction* zu verwenden.

ChangeDimVertical	Ändern der vertikalen Ausdehnung
ChangeDimDown	Verkleinerung der vertikalen Ausdehnung
ChangeDimUp	Vergrößerung der vertikalen Ausdehnung
Delete	Entfernen von Planungselementen
Edit	Ändern von Eigenschaften/Einstellungen, allgemein (s.a. *Electrification*, *Lighting* und *Material*)
Electrification	Ändern von Eigenschaften/Einstellungen, die sich auf die Elektrifizierung beziehen[31]
Flip	Wechsel der Front/Back–Ausrichtung eines Planungselementes (Rotation um die vertikale Achse um 180 Grad)
Lighting	Ändern von Eigenschaften/Einstellungen, die sich auf die Beleuchtung beziehen[32]
Material	Ändern von Materialeigenschaften
OnOff	Ein-/Ausschalten (Umschalten) einer Funktionalität[33]
PosHorizontal	Ändern der horizontalen Position
Pos2Left	Verschiebung nach links
Pos2Right	Verschiebung nach rechts
PosVertical	Ändern der vertikalen Position
PosDown	Verschiebung nach unten
PosUp	Verschiebung nach oben
RotateNY	Rotation um die negative Y–Achse (vertikale Achse, entgegen dem Uhrzeigersinn)
RotateNY90	Rotation um 90 Grad um die negative Y–Achse
RotatePY	Rotation um die positive Y–Achse (vertikale Achse, im Uhrzeigersinn)
RotatePY90	Rotation um 90 Grad um die positive Y–Achse
StartDimChange^{App}	Start einer Aktion vom Typ *DimChange*
Video	Abspielen eines Videos

- Im Feld 6 wird die gewünschte abstrakte Größe des Interaktor-Symbols spezifiziert. Die genauen Abmessungen der Symbole (Piktogramme) je abstrakter Größenstufe werden von den Applikationen festgelegt.

  Folgende Größenstufen sind definiert:

  - **small**
  - **medium**
  - **large**

  Für einen gegebenen Symbol–Typ müssen nicht zwingend Piktogramme in allen Größenstufen vorhanden sein. Die Applikation verwendet dann das Piktogramm in der nächst größeren bzw. nächst kleineren Ausführung.

[31] z.B. Einstellungen, die festlegen, ob und an welcher Position Steckdosen, Kabelführungen, Zuleitungen, Verbindungsleitungen usw. eingeplant werden sollen

[32] z.B. Einstellungen, die festlegen, ob und an welcher Position Leuchten/Lampen welchen Typs eingeplant werden sollen

[33] Da durch das Symbol selber der aktuelle Zustand (an/aus) nicht ersichtlich ist, sollte es nur verwendet werden, wenn durch die Grafik des betroffenen Objektes für den Anwender erkennbar ist, ob die Funktionalität eingeschaltet ist oder nicht (z.B. Licht an/aus, Bildschirm an/aus).

Tabellenname: `SymbolDisplay`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	Interactor	X	ID		X	Bezeichner des Interaktors
2.	HiddenMode	X	BoolExpr		X	Verdeckungsmodus
3.	OffsetType	X	Symbol		X	Art der Angabe des Offsets
4.	Offset	X	Char		X	Offset des Symbols
5.	Direction		ID			Ausrichtung des Symbols
6.	ViewAngle		NumExpr			Öffnungswinkel des Sichtbarkeitsbereiches
7.	OrientationX		ID			Orientierung der X-Achse bei 3D–Symbolen

Anmerkungen:

- In dieser Tabelle werden Position, Ausrichtung und Sichtbarkeitsbereich des Symbols eines Interaktors festgelegt.
- Zu einer Interactor–ID können in der Tabelle mehrere Einträge zur Abbildung unterschiedlicher Positionen mit separaten Sichtbarkeitsbereichen enthalten sein, z.B. zur Realisierung unterschiedlicher Darstellungen für die Frontansicht und die Rückansicht eines Objektes.
  Es liegt in der Verantwortung des Datenanlagers, dafür zu sorgen, dass sich in diesem Fall die Sichtbarkeitsfelder nicht überlappen, damit für den Interaktor, je nach Kamera-Perspektive, nicht u.U. zwei (mehrere) Symbole angezeigt werden.
- Der Modus im Feld 2 gibt an, ob das Interaktor-Symbol durch Objekte verdeckt werden soll, die zwischen der Position des Symbols und der Kameraposition (Anwender/Betrachter) liegen[34]:
  Wenn die Evaluierung des in dem Feld angegebenen Ausdrucks eindeutig *true* ergibt, wird das Symbol verdeckt, anderenfalls nicht[35].
- Die Position des Symbols wird durch einen Offset relativ zur Position des Planungselementes bestimmt, an das der Interaktor gebunden ist.

  Die Angabe des Offsets kann auf zwei Arten erfolgen. Die verwendete Form wird im Feld 3 angegeben:

  Tripel Die 3 Koordinaten (x, y, z) werden in einem Eintrag in der Tabelle `NumTripel` (s. Abschn. 4.2) hinterlegt. Im Feld 4 ist dann die ID dieses Eintrags anzugeben.

  Wird in der Tabelle `NumTripel` kein Eintrag mit der im Feld 4 angegebenen ID gefunden, wird das Symbol an der Position des Planungselementes platziert.

  Expr Die Angabe erfolgt durch einen im Feld 4 hinterlegten Ausdruck, der eine *Sequence* von 3 *Float*–Werten liefert[36]. Der Ausdruck kann u.a. auch Methodenaufrufe beinhalten (Funktion *methodCall()*, s. A.4.15).

  Liefert der im Feld 4 hinterlegte Ausdruck keine Sequenz von 3 Float-Werten, wird das Symbol an der Position des Planungselementes platziert.
- Ist mindestens eines der Felder 5 und 6 leer, so ist das Symbol unabhängig von der Kamera-Perspektive sichtbar[37].
  Um überladene und verwirrende Ansichten zu vermeiden, insbesondere wenn ein Objekt sehr viele Interaktoren besitzt, empfiehlt es sich jedoch, den Sichtbarkeitsbereich der Symbole einzuschränken. In den meisten Fällen ist es z.B. sinnvoll, dass ein Symbol nur dann sichtbar ist, wenn auch die Seite des Objekts sichtbar ist, an der das Symbol angebracht ist.

[34] Das schließt das Objekt ein, an das der Interaktor gebunden ist.
[35] Ist das Feld leer, so ist das Verhalten wie vor der Einführung des Feldes: Wenn *kein* Sichtbarkeitsbereich vorgegeben ist (Felder 5 und 6), wird das Symbol durch davor liegende Objekte verdeckt, anderenfalls nicht.
[36] zu den in Ausdrücken verwendbaren Datentypen siehe Anhang A.2
[37] Allerdings wird es durch Objekte verdeckt, die zwischen der Position des Symbols und der Kameraposition liegen, wenn der Verdeckungsmodus im Feld 2 den Wert *true* hat.

Der *Sichtbarkeitsbereich* eines Symbols wird bestimmt durch einen Richtungsvektor (Feld 5) ausgehend von der Position des Interaktors (Feld 4) und dem Öffnungswinkel relativ zum Richtungsvektor (Feld 6): Wenn das Innere des durch Position des Symbols, Richtungsvektor und Öffnungswinkel bestimmten Trichters, komplett oder teilweise, durch die Kamera mit den aktuellen Kamera-Einstellungen erfasst wird, dann ist das Symbol sichtbar[38].

- Zur Angabe eines Richtungsvektors ist ein Eintrag in der Tabelle `NumTripel` anzulegen und dessen ID im Feld 5 zu hinterlegen. Die Koordinaten des Vektors sind relativ zum lokalen Koordinatensystem des Objektes, an das der Interaktor gebunden ist.

  > Bei einem Symbol an der Frontseite eines Objektes z.B. würde das Tripel `[0.0, 0.0, 1.0]` festlegen, dass das Symbol in Richtung der positiven Z–Achse (nach vorne) ausgerichtet ist.

  Wird in der Tabelle `NumTripel` kein Eintrag mit der im Feld 5 angegebenen ID gefunden, ist das Verhalten wie in dem Fall, dass das Feld leer ist, d.h., das Symbol ist dann immer sichtbar.

- Der Öffnungswinkel im Feld 6 ist in Grad im Bereich von 0 bis 360 anzugeben.

  > Bei einem Symbol an der Frontseite eines Objektes z.B. würde bei einem nach vorne gerichteten Richtungsvektor (Feld 5) der Öffnungswinkel `180` bewirken, dass das Symbol nur sichtbar ist, wenn auch die Frontseite des Objektes sichtbar ist[39].

- Sind die Felder 5 bis 7 nicht leer und wird zu der in dem Feld 7 hinterlegten ID ein Eintrag in der Tabelle `NumTripel` gefunden, wird das Symbol als dreidimensionales Objekt (*3D–Symbol*) behandelt. In allen anderen Fällen wird das Symbol als zweidimensionales Objekt behandelt, welches immer parallel zur Bildebene dargestellt wird (*2D–Symbol*).

  Der Vektor, der durch den im Feld 7 referenzierten Eintrag in der Tabelle `NumTripel` definiert ist[40], legt die X–Achse des lokalen Koordinatensystem des 3D–Symbols fest, dessen Ursprung an der Position des Symbols (Feld 4) liegt. Die Z–Achse dieses Koordinatensystems wird durch den Richtungsvektor[41] (Feld 5) bestimmt und die Y–Achse ergibt sich aus dem Kreuzprodukt von Z- und X–Achse[42].

  Das dem Typ des 3D–Symbols zugeordnete Bild/Piktogramm (s. Feld *SymbolType* in der Tabelle `Interactor` oben) wird dann so platziert und ausgerichtet, dass es in der X-Y–Ebene des 3D–Symbols liegt, der Ursprung (Mitte) des Bildes mit der Position des Symbols übereinstimmt und die X–Achse des Bildes mit der X–Achse des Koordinatensystems des Symbols übereinstimmt, siehe Abb. 2.

  Das Bild eines 2D–Symboles wird so platziert und ausgerichtet, dass es in einer Ebene parallel zur Bildebene (Projektionsebene) liegt, der Ursprung (Mitte) des Bildes mit der Position des Symbols übereinstimmt und die X–Achse des Bildes horizontal nach rechts zeigt.

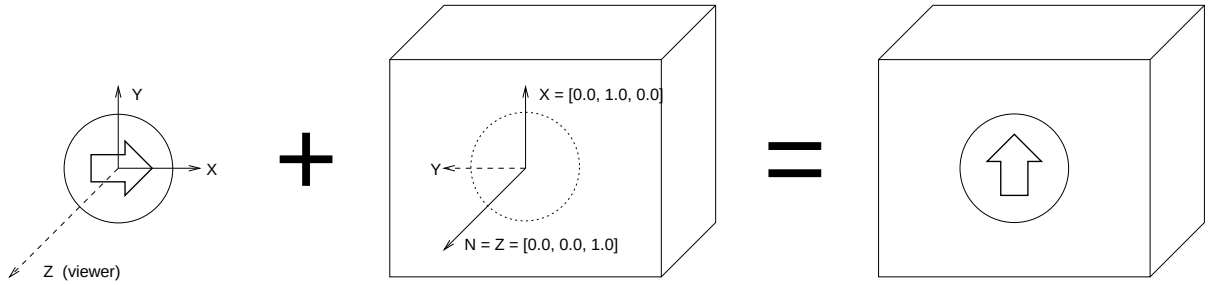
[38] falls der Verdeckungsmodus im Feld 2 den Wert *false* hat oder es nicht durch Objekte verdeckt ist, die zwischen der Position des Symbols und der Kameraposition liegen

[39] In der Praxis sind eher Winkel kleiner 180 Grad sinnvoll.

[40] Koordinaten sind relativ zum lokalen Koordinatensystem des Objektes, an das der Interaktor gebunden ist.

[41] Im Fall eines 3D–Symbols wird der Richtungsvektor auch als *Normale* bezeichnet, da er auch die Ebene definiert, in der die zweidimensionale Abbildung des Piktogramms liegt.

[42] Falls der Vektor für die X–Achse nicht orthogonal zur Normalen ist, führt die Applikation eine Normalisierung des Vektors durch. Der Vektor für die X–Achse darf nicht parallel zur Normalen liegen. Wenn dies doch der Fall sein sollte, wird ein Fehler ausgelöst und das Verhalten der Applikation ist undefiniert.



a) Icon b) Pos./Orientation of interactor symbol c) Mapped symbol icon

Abbildung 2: Abbildung der Piktogramme bei 3D–Symbolen

- Die Länge der durch die Felder 5 und 7 referenzierten Vektoren muß größer als Null sein, ansonsten wird ein Fehler ausgelöst und das Verhalten der Applikation ist undefiniert.

4.7 Aktionen

Tabellenname: `Action`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	Action	X	ID		X	Bezeichner der Aktion
2.	Condition		BoolExpr			Ausführungsbedingung
3.	Type		Symbol		X	Typ der Aktion
4.	Parameter		ID		(X)	Aktionsparameter
5.	Objects		OID_List		(X)	Target-Objekte

Anmerkungen:

- Die Aktion wird ausgeführt (d.h., ist *gültig*), wenn das Feld 2 leer ist oder die Evaluierung des in dem Feld angegebenen Ausdrucks eindeutig *true* ergibt.

  Darüber hinaus können für bestimmte Aktionstypen zusätzliche Bedingungen gelten, die erfüllt sein müssen, damit die Aktion als gültig angesehen wird.
  Diese Bedingungen, falls existent, werden unten bei den einzelnen Aktionstypen aufgeführt.

- Mögliche Typen (Feld 3) sind:

  ActionChoice
  Dies ist keine Aktion im eigentlich Sinne, vielmehr wird hiermit eine Liste von *Optionen* abgebildet, aus der der Anwender eine auswählen kann. Die Auswahl einer Option bewirkt dann die Ausführung von einer oder mehreren Aktionen.

  Die ID im Feld 4 verweist auf die Tabelle `ActionChoice`.
  Diese Tabelle wiederum verweist auf die Tabelle `ActionList`, in der die Liste der auswählbaren Optionen abgebildet ist. Eine Option kann dabei Bedingungen besitzen, die erfüllt sein müssen, damit sie als gültig angesehen wird.

  Eine Aktion vom Typ *ActionChoice* ist nur dann gültig, abgesehen von der Bedingung im Feld 2, wenn es (aktuell) mindestens eine gültige Option gibt.

  CreateObj
  Die Aktion erzeugt ein Objekt.

  Die ID im Feld 4 verweist auf die Tabelle `CreateObj`.

DeleteObj
Die Aktion entfernt die im Feld 5 spezifizierten Objekte.

(Das Feld 4 hat für diesen Aktionstyp keine Bedeutung.)

DimChange
Die Aktion ermöglicht die interaktive Änderung der Dimension(en) des aktiven Objektes.

Die ID im Feld 4 verweist auf die Tabelle `DimChange`.

Message
Die Aktion gibt eine Meldung an den Anwender aus.

Die ID im Feld 4 verweist auf die Tabelle `Message`.

Eine Aktion dieses Typs ist nur für Interaktoren mit dem Symbol–Typ *Attention* erlaubt!

MethodCall
Die Aktion ruft eine OFML–Methode auf.

Die ID im Feld 4 verweist auf die Tabelle `MethodCall`.

Handelt es sich bei der Methode um eine Instanz-Methode, erfolgt der Aufruf auf den im Feld 5 spezifizierten Objekten. Bei Klassen-Methoden hat das Feld 5 keine Bedeutung.

NoAction
Dieser Typ definiert eine *Dummy*–Aktion, zu der keine weiteren Daten angegeben werden müssen (bzw. können) und die immer gültig ist.

Da diese Aktionen keinen wirklichen Effekt haben, sollten sie in ausgelieferten OFML–Daten nicht verwendet werden, sie können jedoch in frühen Phasen der Projektentwicklung nützlich sein, wenn zunächst nur die Interaktorsymbole erstellt und positioniert werden sollen.

PropChange
Die Aktion weist einer Property einen (neuen) Wert oder Status zu.

Die ID im Feld 4 verweist auf die Tabelle `PropChange`.

PropEdit2
Die Aktion führt einen Dialog zur Eingabe bzw. Auswahl von Propertywerten aus.

Die ID im Feld 4 verweist auf die Tabelle `PropEdit2`.

Die aus Feld 5 resultierende Objektmenge darf nur genau ein Objekt enthalten, ansonsten hat die Aktion keinen Effekt.

SelectObj
Die Aktion selektiert das im Feld 5 spezifizierte Objekt.

(Das Feld 4 hat für diesen Aktionstyp keine Bedeutung.)

Die aus Feld 5 resultierende Objektmenge darf nur genau ein Objekt enthalten, ansonsten hat die Aktion keinen Effekt.

Achtung:
Die Aktion hat einen Wechsel des aktiven Objekts zur Folge (siehe Objektkategorie *Self*, Abschn. 4.9). Die Abarbeitung der Aktionsliste eines Interaktors (s.a. Abschn. 4.6) wird deswegen nach dieser Aktion abgebrochen! Aktionen dieses Typs sollten also immer die letzte Aktion in der Aktionsliste eines Interaktors sein.

ShowMedia
Die Aktion zeigt dem Anwender einen (externen) Medieninhalt an.

Die ID im Feld 4 verweist auf die Tabelle `ExtMedia`.

- In Feldern vom Typ *ID_List*, in denen eine Liste von Aktionsidentifikatoren angegeben werden kann, darf es nur eine Aktion geben, die eine Interaktion mit dem Nutzer beinhaltet[43].
  Bezüglich dieser Aktionen gelten des Weiteren die folgenden Einschränkungen bzw. Festlegungen:
  - Aktionen der Typen *DimChange*, *Message* und *ShowMedia* müssen die einzige Aktion in der Liste sein. (Diese Aktionen werden immer ausgeführt, d.h., eine eventuell angegebene Ausführungsbedingung wird ignoriert.)
  - Aktionen des Typs *PropEdit2* müssen die letzte Aktion in der Liste sein.
  - Wird bei Aktionen des Typs *ActionChoice* die Interaktion (ohne Auswahl) durch den Nutzer abgebrochen, werden evtl. nachfolgende Aktionen nicht ausgeführt.

- Im Feld 5 werden die Objekte spezifiziert, für die die Aktion auszuführen ist.

  Für die Aktionstypen *ActionChoice*, *CreateObj*, *DimChange*, *Message* und *ShowMedia* hat das Feld keine Bedeutung[44]. Für die anderen Aktionstypen gilt folgendes:

  Handelt es sich bei einer OID um einen einfachen Identifikator (siehe Feldtyp *OID*), wird mit dem Identifikator wird auf die Tabelle `Object` zugegriffen, um die betreffende Objektmenge zu ermitteln.

  Bei hierarchischen Namen ergibt sich die Gesamtmenge der betroffenen Objekte aus dem Produkt der Mengen auf den einzelnen Hierarchie-Ebenen.

  Die Aktion wird für jedes der spezifizierten Objekte in der angegebenen Reihenfolge der OID's ausgeführt. Wenn eine OID mehr als ein Objekt referenziert, ist die Reihenfolge der Ausführung innerhalb dieser Menge undefiniert.

4.8 Die Tabellen für die Aktionsparameter

4.8.1 Aktionsauswahl

Zur Abbildung von Aktionsauswahlen werden 2 Tabellen benötigt: In der Tabelle `ActionChoice` werden die allgemeinen Eigenschaften der Aktionsauswahl beschrieben (z.B. das Erscheinungsbild). Des Weiteren wird in dieser Tabelle auf die Liste der wählbaren Optionen verwiesen, welche in der Tabelle `ActionList` hinterlegt wird[45]. In dieser Tabelle erfolgt dann auch die Zuordnung von Bezeichnungen und/oder Bildern zu den Optionen.

Tabellenname: `ActionChoice`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	Title		ID			Text-ID für den Dialog-Titel
3.	ViewType		Symbol		X	Typ/Art der Darstellung
4.	Argument		(Char)		(X)	(bedingter) Parameter für die Art der Darstellung
5.	ListID		ID		X	ID der Aktionsliste

Die ID im Feld 2 verweist auf die Tabelle `Text`.
Wenn das Feld leer ist oder zu der ID kein Text ermittelt werden kann, wird der Auswahl-Dialog ohne Titel angezeigt.

[43] betrifft: *ActionChoice*, *DimChange*, *Message*, *PropEdit2* und *ShowMedia*
[44] Die möglichen Vaterobjekte beim Aktionstyp *CreateObj* werden in der Parametertabelle `CreateObj` hinterlegt.
[45] Die Tabelle sollte besser `ChoiceList` heißen. Für eine einfacherere Migration wird aber weiter der bisherige Name verwendet, der eingeführt wurde, als zu einer Option nur eine Aktion hinterlegt werden konnte.

Aktuell werden 2 Arten der Darstellung (Feld 3) unterstützt:

List In dieser Darstellung werden die Auswahloptionen in einer Liste untereinander dargestellt, wobei jede Option durch ein (optionales) Bildchen links und einen Text rechts davon repräsentiert wird.

Die Größe der Bildchen sollte sich an der Größe für die Materialbilder in Eigenschaftseditoren orientieren (siehe [dsr]), also 50 x 18 Bildpunkte (Breite x Höhe). Das ist aber nicht zwingend. Idealerweise sollten die Bildchen für eine gegebene Aktionsauswahl jedoch dieselbe Größe haben.

Ist für mindestens eine Option ein Bildchen hinterlegt, so bleibt bei den Optionen ohne hinterlegtes Bildchen die dafür vorgesehene Fläche leer, d.h. die Texte aller Optionen sind linksbündig.

In dieser Darstellungsform sollte für eine Option immer ein Text vorhanden sein, da die Bildchen alleine aufgrund der kleinen Größe in der Regel nicht aussagekräftig sind.

Das Feld 4 ist für diese Darstellungsform nicht relevant.

Tile In dieser Darstellung werden die Auswahloptionen in gleich großen Kacheln neben- und/oder untereinander dargstellt. Diese Form ist für die Darstellung von Optionen mit größeren Bildern vorgesehen.

Eine Kachel besteht aus dem Bild und einem (optionalen) Text darunter oder rechts daneben.

Im Feld 4 wird die gewünschte Kachelgröße angegeben, wobei sich diese rein auf die Größe des Bildes bezieht. (Mit Text ist die tatsächlich angezeigte Kachel entsprechend größer.)

Aktuell werden folgende Kachelgrößen unterstützt (Bildpunkte, Breite x Höhe):

small 50 x 50

medium 100 x 100

large 200 x 200

Es wird erwartet, dass die Bilddateien für normalauflösende Displays in genau der Größe (in Pixeln) vorliegen, die durch die im Feld 4 angegebene Kachelgröße bestimmt ist.

Die o.g. Bildmaße sind Angaben in (logischen) Bildpunkten. Für eine gute Darstellung auf hochauflösenden Displays sind Bilddateien mit einer entsprechend größeren Auflösung (Pixel) bereitzustellen. Details dazu siehe Tabelle `Image` (Abschn. 4.11).

Eine Option ohne Text *und* Bild wird nicht angezeigt.

Die ID im Feld 5 verweist auf die folgende Tabelle.

Tabellenname: `ActionList`
Pflichttabelle: bedingt (ja, wenn Tabelle `ActionChoice` vorhanden ist)

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID der Aktionsliste
2.	Position	X	Int		X	Position der Aktion in der Liste
3.	Condition		BoolExpr			Gültigkeitsbedingung
4.	Actions		ID_List		X	Aktionen
5.	TextID		ID			ID des Textes
6.	ImageID		ID			ID des Bildes

Anmerkungen:

- Ein Eintrag in der Liste (eine Option) ist *gültig*, wenn das Feld 3 leer ist oder wenn die Evaluierung des in dem Feld angegebenen Ausdrucks eindeutig *true* ergibt.
- Die IDs im Feld 4 verweisen auf die Tabelle `Action`.

  Die referenzierten Aktionen dürfen selber auch den Typ *ActionChoice* haben, d.h. es wird eine Verschachtelung von Aktionen des Typs *ActionChoice* unterstützt[46].

[46] Die konkrete Umsetzung in der Nutzeroberfläche kann von Anwendung zu Anwendung verschieden sein.

Darüberhinaus sollten keine Aktionen verwendet werden, die selber wieder einen Nutzerdialog beinhalten.

Für die Abarbeitung der Aktionen gelten dieselben Festlegungen wie für die Aktionen eines Interaktors (s. 4.6), konkret:

Die Aktionen werden in der Reihenfolge der IDs ausgeführt. Dabei wird eine Aktion übersprungen, wenn deren Ausführungsbedingung aktuell nicht erfüllt ist, oder wenn sie anderweitig *ungültig* ist.

Eine Option ist nur dann gültig, wenn es (aktuell) mindestens eine gültige Aktion gibt.

Tritt beim Zugriff auf die Parameter der referenzierten Aktionen in der OAP–Datenbank ein Fehler auf, oder schlägt die Ausführung einer der referenzierten Aktionen fehl, wird die Abarbeitung der Aktionsliste abgebrochen und die umschließende *ActionChoice*–Aktion schlägt fehl!

- Die ID im Feld 5 verweist auf die Tabelle `Text` (Abschn. 4.10). In dieser Tabelle können sprachspezifische Bezeichnungen für die Aktion/Option hinterlegt werden.
- Die ID im Feld 6 verweist auf die Tabelle `Image` (Abschn. 4.11). In dieser Tabelle kann ein Bild (Icon) zur Veranschaulichung der Aktion/Option hinterlegt werden[47].

  Bei Bedarf können in der Tabelle `Image` auch sprachspezifische Bilddateien referenziert werden.

4.8.2 `PropChange`

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	Type		Symbol		X	Typ/Art der Änderung
3.	Property		Symbol		X	Property–Key
4.	Value		Char		X	Ausdruck für den Wert der Änderung

Anmerkungen:

- Folgende Änderungsarten sind definiert:

  Value Der Property wird ein (neuer) Wert zugewiesen.

  Der Ausdruck[48] im Feld 4 muß einen Wert liefern, der zum Datentyp der Property passt.

  Visibility Die Sichtbarkeit der Property wird geändert.

  Der Ausdruck im Feld 4 muß einen booleschen Wert liefern (s. Feldtyp *Bool*).
  Der Wert legt fest, ob die Property sichtbar sein soll oder nicht.

  Editability Die Editierbarkeit der Property wird geändert.

  Der Ausdruck im Feld 4 muß einen booleschen Wert liefern (s. Feldtyp *Bool*).
  Der Wert legt fest, ob die Property editierbar sein soll oder nicht.

- Im Feld 3 wird der Schlüssel der zu ändernden Property (ohne das vorangestellte `@`–Zeichen) angegeben.

  Die Änderung wird für alle von der Aktion betroffenen Objekte durchgeführt, welche die diese Property besitzen.

[47] wobei zu einem Bild mehrere Bilddateien mit entsprechenden Auflösungen für verschieden auflösende Displays zu hinterlegen sind (Details siehe Tabelle `Image`)

[48] zu Ausdrücken s. Anhang A

4.8.3 PropEdit2

Tabellenname: `PropEdit2`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	Title		ID			Text-ID für den Editor-Titel
3.	Properties		ID		(X)	ID für `PropEditProps`
4.	Classes		ID		(X)	ID für `PropEditClasses`

Anmerkungen:

- Die ID im Feld 2 verweist auf die Tabelle `Text`.
  Wenn das Feld leer ist oder zu der ID kein Text ermittelt werden kann, wird der Editor–Dialog ohne Titel angezeigt[49].
- Die zu verwendenden Properties bzw. Property-Klassen (wenn alle Properties einer Klasse in Frage kommen) werden in den Tabellen `PropEditProps` bzw. `PropEditClasses` spezifiziert (s.u.).
  Die ID für den Zugriff auf diese Tabellen wird in den Feldern 3 bzw. 4 hinterlegt. (Nur eines der Felder darf leer sein.)
- Properties aus Klassen, die über die Tabelle `PropEditClasses` referenziert werden, müssen/sollten nicht extra noch in der Tabelle `PropEditProps` angegeben werden.

Tabellenname: `PropEditProps`
Pflichttabelle: bedingt (ja, wenn Tabelle `PropEdit2` vorhanden ist und Referenzen im Feld *Properties* enthält)

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	Property	X	Symbol		X	Property-Key
3.	Condition		BoolExpr			Anwendungsbedingung
4.	StateRestr		Symbol		X	Einschränkung hinsichtlich Property-Status

Property-Keys (Feld 2) sind ohne das vorangestellte `@`–Zeichen anzugeben.

Zu den Feldern 3 und 4 siehe unten.

Tabellenname: `PropEditClasses`
Pflichttabelle: bedingt (ja, wenn Tabelle `PropEdit2` vorhanden ist und Referenzen im Feld *Classes* enthält)

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	PropClass	X	Char		X	Name der Klasse
3.	Condition		BoolExpr			Anwendungsbedingung
4.	StateRestr		Symbol		X	Einschränkung hinsichtlich Property-Status

Eine Property bzw. eine Klasse kommt prinzipiell nur dann zur Anwendung im Dialog, wenn das Feld 3 (*Condition*) leer ist oder wenn die Evaluierung des in dem Feld angegebenen Ausdrucks eindeutig *true* ergibt.

[49] In diesem Fall soll die Applikation auch nicht etwa die Bezeichnung der Property als Fallback heranziehen (insofern nur eine Property involviert ist).

Hinweis:
Da die Bedingungen während des Dialogs nach einer Property-Änderung *nicht* (erneut) ausgewertet werden, dürfen sich diese nicht auf Properties beziehen, deren Werte sich während des Dialogs selber ändern können!
Zur Behandlung der Abhängigkeit der Sichtbarkeit einer Property von anderen Properties im Allgemeinen und im Speziellen von Properties, die im Dialog selber zur Anwendung kommen, sollte im nachfolgenden Feld 4 ein entsprechender Wert (ungleich `None`) angegeben werden.

Im Feld 4 (*StateRestr*) wird angegeben, ob die Property bzw. die Properties der Klasse abhängig vom aktuellen Property-Status im Dialog angezeigt werden sollen. (Das Feld wird nur für Properties bzw. Klassen ausgewertet, die gemäß Feld 3 gültig sind.)

Aktuell sind folgende Werte für das Feld *StateRestr* definiert:

None Keine Einschränkung, d.h., die Property bzw. die Properties der Klasse soll(en) unabhängig von aktuellen Status angezeigt werden.

Visible Die Property bzw. die Properties der Klasse soll(en) nur dann angezeigt werden, wenn sie aktuell sichtbar ist/sind[50].

VisibleEditable Die Property bzw. die Properties der Klasse soll(en) nur dann angezeigt werden, wenn sie aktuell sichtbar **und** editierbar ist/sind.

Kommt aufgrund der Anwendungsbedingungen nur eine Property für die Verwendung im Dialog in Frage, hat das Feld *StateRestr* für diese Property keine Bedeutung, d.h., die Property wird unabhängig von ihrem aktuellen Status verwendet und kann vom Anwender geändert werden[51].

Kommt unter Berücksichtigung der Bedingungen und des Property-Status nur eine Property zur Anzeige, wird im Editor–Dialog nur das Eingabefeld bzw. die Auswahlliste dieser Property angezeigt. (Die Anzeige der Property-Bezeichnung entfällt.) Der Dialog wird beendet, sobald der Anwender einen Wert ausgewählt bzw. die Eingabe bestätigt hat.
Falls mehrere Properties editiert werden, bietet die Applikation eine geeignete GUI–Technik an, mit der der Anwender den Dialog explizit beenden kann/muss. Die Grafik des betreffenden Objekts wird dabei aber bei jeder Property-Änderung aktualisiert, d.h. nicht erst bei Beendigung des Dialogs.

Für ein optimales Erscheinungsbild des Dialogs sollten die Materialbilder für die Property-Werte auch in der großen Variante angelegt werden (s. [dsr], Abschn. 6.4.2).

4.8.4 `DimChange`

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	Dimension	X	Symbol		X	betroffene Dimension
3.	Condition		BoolExpr			Anwendungsbedingung
4.	Separate		BoolExpr		X	Nur separat änderbar?
5.	ThirdDim		BoolExpr		X	Als dritte Dimension verwenden?
6.	Property		Symbol		X	anzuwendende Property
7.	Multiplier		NumExpr		X	Faktor zur Umrechung in m
8.	Precision		NumExpr		X	Genauigkeit in m

Anmerkungen:

- Wie schon in Abschn. 4.7 festgelegt, müssen Aktionen vom Typ *DimChange* immer die erste in der Liste der Aktionen eines Interaktors sein. Das Symbol des Interaktors muss den Typ *StartDimChange* haben[52].

[50] Diese Einschränkung ist dann sinnvoll bzw. erforderlich, wenn die im Dialog verwendeten Properties Abhängigkeiten untereinander aufweisen.
[51] auch wenn sie an sich nicht editierbar ist
[52] Andere Typen werden ignoriert und durch *StartDimChange* ersetzt.

- Aktionen vom Typ *DimChange* werden aktuell nur für Objekte auf der obersten Planungsebene unterstützt.

- Im Feld 2 wird die Achse der betroffenen Dimension spezifiziert.

  Die möglichen Werte sind `X`, `Y`, `Z`, `PX`, `PY`, `PZ` sowie `NX`, `NY` und `NZ`:

  – Bei `X`, `Y` und `Z` darf die Änderung auf beiden Seiten des Objektes erfolgen.
  – Bei `PX`, `PY` sowie `PZ` darf die Änderung nur an der Seite in positiver Richtung der Achse und bei `NX`, `NY` sowie `NZ` nur an der Seite in negativer Richtung der Achse erfolgen.
  – Eine Änderung an der Seite in negativer Richtung der Achse bewirkt eine entsprechende Umpositionierung des Objektes.
  – Pro Dimension darf nur eine der 3 Varianten (der möglichen Änderungsrichtungen) verwendet werden, oder mittels der Bedingung im Feld 3 (s.u.) muss sichergestellt werden, dass zum Zeitpunkt der Auswertung nur eine der 3 Varianten gültig ist. (Anderenfalls findet die betreffende Dimension keine Anwendung.)

- Die im Feld 2 angegebene Dimension wird angewendet, wenn das Feld 3 leer ist oder die Evaluierung des in dem Feld angegebenen Ausdrucks eindeutig *true* ergibt.

  Ist nach Auswertung der Bedingungen aller Einträge für die Aktion (Feld 1) mehr als eine Dimension betroffen, ist es der Applikation überlassen, ob und wie sie die Interaktion realisiert. (Dies kann u.U. von der aktuellen Ansicht des Nutzers abhängig sein.) Das Verhalten der Applikation in dieser Situation kann jedoch in gewissem Rahmen auch durch die Angaben in den Feldern 4 und 5 beeinflusst werden.

- Sollen 2 (oder 3) Dimensionen gleichzeitig geändert werden können, dürfen die Wertemengen und -bereiche der betreffenden Properties nicht voneinander abhängig sein! (Ansonsten kann es zu einem unerwarteten bzw. verwirrenden Feedback-Erlebnis für den Anwender kommen.)
  Ist diese Voraussetzung nicht erfüllt, muss die Evaluierung des im Feld 4 (*Separate*) angegebenen Ausdrucks für die betreffenden Dimensionen *true* ergeben.
  Die Applikationen rufen dann nach abgeschlossener Änderung einer Dimension[53] die Wertemengen und -bereiche der betreffenden Properties erneut ab, um auf mögliche Änderungen aufgrund der Abhängigkeiten zu reagieren.

- Nicht alle Applikationen bzw. Applikationsmodi erlauben die gleichzeitige Änderung aller drei Dimensionen. Ist seitens der Daten eine gleichzeitige Änderung möglich (siehe Feld *Separate*), sollte im Feld 5 (*ThirdDim*) angegeben werden, welche der 3 Dimensionen bei Bedarf separat zu ändern ist. Die Evaluierung des im Feld angegebenen Ausdrucks muss dazu *true* ergeben.
  Ist nicht genau eine Dimension als *dritte Dimension* deklariert, ist das Verhalten der Applikation undefiniert.

- Im Feld 6 wird die Property spezifiziert, die zur Realisierung der Änderung der betroffenen Dimension zu verwenden ist.

  Der Property-Key ist ohne das vorangestellte `@`–Zeichen anzugeben.

  Die Property muß folgende Bedingungen erfüllen[54]:

  – Der Datentyp der Property muß numerisch sein (`N`) oder eine symbolische Auswahlliste (`Y`, `YS`), siehe [property].

    Im Fall einer symbolischen Auswahlliste muss die Klasse der OFML–Instanz folgende Methode implementieren:

    symbolicPropValue2Float(pPKey(Symbol), pPValue(Symbol)) → Float

    > Die Methode liefert den numerischen Wert in *Meter*, der dem angegebenen (symbolischen) Wert der spezifizierten Property der impliziten Instanz entspricht.

    Die Aussagen im folgenden beziehen sich auf die nativen Werte im Fall einer numerischen Property bzw. auf die via *symbolicPropValue2Float()* ermittelten numerischen Werte im Fall einer symbolischen Auswahlliste.

[53] bei noch nicht abgeschlossener Aktion
[54] Ist eine davon nicht erfüllt, schlägt die Aktion fehl.

  - Die Werte der Property müssen positiv sein.
  - Die Property muß entweder *einen* geschlossenen Wertebereich definieren (im Fall einer numerischen Property) oder eine Auswahlliste von Einzelwerten (Wertemenge).
    Ist beides definiert, werden im Dialog nur diejenigen Werte aus der Auswahlliste verwendet, die im Wertebereich liegen.

- Im Feld 7 ist der Faktor anzugeben, der zur Umrechnung von Property-Werten in *Meter* anzuwenden ist.

  Im Fall einer symbolischen Auswahlliste wird der Faktor `1.0` angenommen[55].

- Im Feld 8 ist die Genauigkeit der Property-Werte (in *Meter*) anzugeben.

Vor der Zuweisung eines durch den Dialog ermittelten (neuen) Wertes für die betroffene Dimension an die im Feld 6 angegebene Property führt die Applikation somit folgende Berechnungen (in dieser Reihenfolge) durch:

1. Rundung des Wertes auf die im Feld 8 spezifizierte Genauigkeit.
2. Division des im ersten Schritt ermittelten (gerundeten) Wertes durch den im Feld 7 angegebenen Faktor.
3. Rundung des im zweiten Schritt ermittelten Wertes auf die Genauigkeit der Property-Werte laut Property-Definition[56].
   (Dieser Schritt ist erforderlich, falls der Faktor im Feld 7 keine Zehnerpotenz ist, z.B. bei Property-Werten in Zoll.)

Im Fall einer symbolischen Auswahlliste wird zu dem so ermittelten numerischen Wert der zugehörige symbolische Wert bestimmt und dieser an die Property zugewiesen.

4.8.5 `CreateObj`

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	Parent		OID		X	übergeordneter Artikel
3.	ArtSpecMode		Symbol		X	Art der Angabe des zu erzeugenden Artikels
4.	Package		OFML		(X)	OFML–Package
5.	ArticleID		Char		(X)	Artikelnummer
6.	VarCode		Char			OFML-Variantencode
7.	PosRotMode		Symbol		X	Art der Angabe von Position/Rotation
8.	PosRotArg1		(Char)		(X)	Argument 1 für Position/Rotation
9.	PosRotArg2		(Char)		(X)	Argument 2 für Position/Rotation
10.	PosRotArg3		(Char)		(X)	Argument 3 für Position/Rotation

Anmerkungen:

- Die Objekt–ID im Feld 2 darf nur ein Objekt referenzieren, ansonsten findet keine Artikelerzeugung statt.

- Der Modus in Feld 3 spezifiziert, wie die Angabe des zu erzeugenden Artikels erfolgt. Folgende Modi werden unterstützt:

  Explicit Die Angaben erfolgen explizit in den Feldern 4-6. Soll eine bestimmte Variante erzeugt werden, muss im Feld 6 der (möglicherweise teilbestimmte) OFML–Variantencode angegeben werden.

[55] da die Methode *symbolicPropValue2Float()* die numerischen Werte in *Meter* liefert
[56] Im Fall einer symbolischen Auswahlliste werden 3 Dezimalstellen angenommen.

Self Es wird ein Artikel mit derselben Artikelnummer und derselben Konfiguration (OFML-Variantencode) erzeugt wie das Objekt, an das der Interaktor gebunden ist, für den die Aktion ausgelöst wurde[57].

- Der Modus in Feld 7 spezifiziert, wie die Angabe von Position und Rotation erfolgt. Folgende Modi werden unterstützt:

  DataDefined
  Position und Rotation werden durch die OFML–Daten festgelegt.

  Die Applikation ermittelt Position und Rotation für den neuen Artikel mittels Aufruf der Methode *checkAdd()* (OFML–Schnittstelle *Complex*) auf der OFML–Instanz des übergeordneten Artikels.

  Im Feld 8 kann mittels einer Objekt–ID ein an die Methode *checkAdd()* zu übergebendes Referenzobjekt spezifiziert werden (welches ein unmittelbarer Unterartikel des übergeordneten Artikels sein muss).

  Zusätzlich kann im Feld 9 der Schlüssel eines OFML–Anfügepunktes des Referenzobjektes angegeben werden (ohne das vorangestellte `@`–Zeichen), welcher bei der Platzierung bevorzugt verwendet werden soll[58].

4.8.6 `MethodCall`

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	Type		Symbol	8	X	Art des Aufrufs
3.	Context		OFML		X	Aufruf-Kontext
4.	Method		Char		X	Name der Methode
5.	Arguments		Char			Argumente

Es werden 2 Arten von Methodenaufrufen unterstützt. Die jeweils vorliegende Art muss im Feld 2 spezifiziert werden:

Instance Die Methode wird auf einer OFML–Instanz (Objekt) aufgerufen.

Im Feld 3 ist dazu der vollqualifizierte Bezeichner des OFML–Typs oder der OFML–Schnittstelle anzugeben, für die die Methode definiert ist, und im Feld 4 der Name der Methode.

Der Aufruf der Methode erfolgt auf jedem Target-Objekt der Aktion (s. Feld *Objects* in der Tabelle `Action`, Abschn. 4.7), dessen Klasse von dem im Feld 3 spezifizierten Typ abstammt bzw. die dort spezifizierte Schnittstelle implementiert[59].

Class Es wird eine (statische) Klassen-Methode aufgerufen.

Im Feld 3 ist dazu der vollqualifizierte Bezeichner der Klasse (OFML–Typ) anzugeben und im Feld 4 der Name der Methode.

Im Feld 5 können optional Argumente für den Methodenaufruf angegeben werden. Mehrere Argumente sind durch ein Komma zu trennen. Jedes Argument wird als OAP–Ausdruck angegeben (s. Anhang A). U.a. können dabei auch *Platzhalter* verwendet werden (s. A.4.17).

[57] D.h., die im Modus *Explicit* in den Feldern 4-6 hinterlegten Angaben werden hier durch die Applikation vom aktiven Objekt abgefragt.

[58] bezieht sich auf die Methode *setActiveAttPt()* der OFML–Schnittstelle *AttachPts*

[59] Insofern die Artikel in der Applikation nicht bereits durch eine OFML–Instanz repräsentiert sind, muß die Applikation zu diesem Zweck eine entsprechende temporäre OFML–Instanz erzeugen.

4.8.7 `Message`

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	ArgType		Symbol		X	Art des Arguments
3.	Argument		ID		X	Argument

Anmerkungen:

- Der Text für die Nachricht kann entweder in der Text–Tabelle hinterlegt werden oder über einen Methodenaufruf bereitgestellt werden.
  (Letzteres ist sinnvoll, wenn der Text situationsabhängig aus Bausteinen zusammengesetzt werden muss.)

  Im Feld 2 wird angegeben, welche der beiden Varianten verwendet wird:

  Text Die ID im Feld 3 verweist auf die Tabelle `Text` (Abschn. 4.10).

  Method Die ID im Feld 3 verweist auf die Tabelle `Action` (Abschn. 4.7).

  Die referenzierte Aktion muss vom Typ *MethodCall* sein und das Ergebnis des Methodenaufrufs muss vom OFML–Typ *String* sein.
  Ist das nicht der Fall, wird die ID selber als Nachricht ausgegeben.

  Es wird erwartet, dass die Methode den Text in der Sprache liefert, die aktuell für Produktdatentexte der OFML–Serie des aktiven Objektes und/oder des Target-Objektes zu verwenden ist (siehe Funktion *getPDLanguage()* der OFML–Schnittstelle *Article* in [article]).

  Der von der Methode gelieferte String muss entweder aus einer String–Ressource–Datei gelesen werden oder in US-ASCII kodiert sein.

- Zur Formatierung stehen folgende Möglichkeiten zur Verfügung:
  - Die Zeichenfolge `\n` bewirkt einen Zeilenumbruch.
- Der Dialog wird beeendet, wenn der Anwender neben das Dialogfenster oder auf den OK–Button „klickt“.

4.8.8 `ExtMedia`

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	ID des Parametersatzes
2.	Type		Symbol		X	Media-Typ
3.	Media		String		X	Media-ID

Anmerkungen:

- Der Zugriff auf externe Medieninhalte erfolgt grundsätzlich *nicht* mittels der direkten Angabe einer URL, sondern über eine ID (Feld 3).
- Folgende Medientypen (Feld 2) sind definiert:

  PIM ein über das PIM[60] bereitgestellter Medieninhalt

  Im Feld 3 muss die ID angegeben werden, mittels derer über die PIM–Schnittstelle auf den Medieninhalt zugegriffen werden kann.
  (Dieser Typ wird aktuell noch nicht unterstützt.)

[60] Product Information Management

YouTube ein von YouTube gehostetes Video

Im Feld 3 muss die YouTube Video-ID angegeben werden.
Diese ist aus der URL des Videos zu entnehmen. (Z.B., bei der URL `https://www.youtube.com/watch?v=k-W5A-mvphg` ist die Video-ID `k-W5A-mvphg`.)

- Es ist den Applikationen überlassen, ob die Anzeige des Medieninhaltes in einem Fenster (Dialog) der Applikation selber oder über eine externe App (z.B. Browser) erfolgt.

4.9 Objektdefinitionen

Ein Eintrag in dieser Tabelle referenziert ein spezielles Objekt oder eine Menge von Objekten. Die Referenzierung erfolgt primär durch Angabe einer Objektkategorie, die eine definierte Menge von Objekten beschreibt. Bei einigen Kategorien, die ein Menge von mehr als einem Objekt beschreiben, kann die Menge durch Kategorie-spezifische Argumente eingeschränkt werden.

Tabellenname: `Object`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	Objekt-ID
2.	Category		Symbol		X	Objektkategorie
3.	Argumen1		Char			Argument 1
4.	Argumen2		Char			Argument 2
5.	Argumen3		Char			Argument 3

Anmerkungen:

- Die ID in Feld 1 ist (möglicherweise der einzige) Teil eines hierarchischen Objektidentifikators vom Feldtyp *OID*. Objektidentifikatoren (OID's) spezifizieren die Target-Objekte von Aktionen (siehe Feld *Objects* in der Tabelle `Action`, Abschn. 4.7) oder Argumente von spezifischen Aktionstypen.
- Folgende Objektkategorien (Feld 2) sind definiert:

  Self Referenziert das aktive Objekt bei einer *AttachAction* bzw. einer *DetachAction* eines Anfügebereich-Paares (s. Tabelle `AttAreaMatch`, Abschn. 4.5) oder das Objekt, an das der Interaktor gebunden ist, für den die Aktion ausgelöst wurde.

  Die Argument-Felder 3-5 haben keine Bedeutung.

  ParentArticle
  Referenziert den unmittelbar übergeordneten Artikel von *Self*.
  Die resultierende Objektmenge ist leer, wenn *Self* kein Unterartikel ist.

  Die Argument-Felder 3-5 haben keine Bedeutung.

  TopArticle Referenziert den in der Hierarchie am höchsten stehenden übergeordneten Artikel von *Self*.
  Die resultierende Objektmenge ist leer, wenn *Self* kein Unterartikel ist.

  Die Argument-Felder 3-5 haben keine Bedeutung.

  MethodCall Die Objekte werden durch einen Methodenaufruf ermittelt.

  Diese Objektkategorie ist nicht für das Target-Objekt einer Aktion erlaubt, deren ID als Argument eines Aufrufs der Funktion *methodCall()* (s. A.4.15) verwendet wird!

  Im Feld 3 ist die ID einer Aktion vom Typ *MethodCall* anzugeben.
  Im Fall einer Instanz-Methode ist dabei zur Festlegung des Target-Objektes dieser Aktion nur eine Objektdefinition mit einer der Objektkategorien *Self*, *ParentArticle* und *TopArticle* erlaubt.

Es wird erwartet, dass die Methode entweder eine Referenz auf eine OFML–Instanz oder eine (nicht-leere) Sequenz (Vector, List) von Referenzen auf OFML–Instanzen liefert. Die OFML–Instanzen müssen dabei (je) einen Artikel repräsentieren[61].
Die Methode darf keine Nebenwirkungen haben, welche eine Aktualisierung der in verschiedenen Teilen der Anwendung gespeicherten Informationen über OFML–Instanzen erfordern würden[62].

Die Argument-Felder 4 und 5 haben keine Bedeutung.

Die Ausführung der Aktion, in welcher die Objektdefinition verwendet wird, schlägt in folgenen Fällen fehl:

- Die im Feld 3 angegebene Aktion hat nicht den Typ *MethodCall*.
- Die im Feld 3 angegebene Aktion spezifiziert eine Instanz-Methode, aber es ist mehr als ein Target-Objekt angegeben oder die Objektdefinition für das Target-Objekt verwendet nicht eine der Objektkategorien *Self*, *ParentArticle* und *TopArticle*.
- Die Methode liefert keine OFML–Instanz oder mehr als eine OFML–Instanz, wo genau ein Objekt erwartet wird.

4.10 Texte

In dieser Tabelle werden sprachspezifische Texte hinterlegt, die für Aktionen der Typen *ActionChoice*, *Message* und *PropEdit2* benötigt werden.

Tabellenname: `Text`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	Text-ID
2.	Language	X	Lang	5		Sprachschlüssel
3.	Text		Text		X	Text-Inhalt

Zur Verwendung und Auswertung des Sprachschlüssels (Feld 2) siehe 3.3.

Insofern bei den oben genannten Aktionen nicht anders angegeben, sind in dem Text die aus OFML [ofml] bekannten Escape-Sequenzen für Sonderzeichen *nicht* erlaubt!

Existiert die Tabelle nicht, ist das Verhalten der Applikation undefiniert. In der Regel wird die betreffende Aktion dann fehlschlagen.
Existiert die Tabelle, aber kann kein Text zu einer ID ermittelt werden, wird die ID selber als Text verwendet.

4.11 Bilder

Tabellenname: `Image`
Pflichttabelle: nein

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	ID	X	ID		X	Image-ID
2.	Language	X	Lang	5	X	Sprachschlüssel
3.	DPR	X	Int		X	Device-Pixel-Ratio
4.	File		Char		X	Dateiname/pfad

[61] zu den Begriffen s.a. Abschn. 2.2
[62] s.a. Anmerkungen zur Funktion *methodCall()* in A.4.15

Anmerkungen:

- Zur Verwendung und Auswertung des Sprachschlüssels (Feld 2) siehe 3.3.

  Kann kein Bild zu einer ID ermittelt werden, ist das Verhalten der Applikation undefiniert. In der Regel wird die betreffende Aktion dann fehlschlagen.

- Die *Device-Pixel-Ratio* (Feld 3) ist ein Begriff aus dem Web Design: Sie gibt an, wieviele physische Pixel des Ausgabegerätes – pro Dimension – zur Darstellung eines (logischen) Bildpunktes (der Webseite) verwendet werden. Bei einer DPR von 2 werden also 4 Pixel zur Darstellung eines Bildpunktes verwendet. Normalauflösende Displays haben eine DPR von 1, hochauflösende Displays von Smartphones und Tablets eine DPR von 2 (und mehr).

  Zur Darstellung eines Bildes mit 100x100 Bildpunkten (z.B. eine Kachel der Größe *medium* bei der Darstellungsform *Tile* von Aktionen des Typs *ActionChoice*) wird bei einem hochauflösenden Display mit DPR 2 also eine Bilddatei der Größe 200x200 Pixel erwartet[63]. Würde stattdessen eine Bilddatei mit nur 100x100 Pixeln bereitgestellt, würde das Image auf 200x200 Pixel hochskaliert werden[64], was zu Qualitätsverlusten führen würde.

  Im Rahmen von OAP wird deswegen gefordert, dass neben der Bilddatei für normale Auflösung (DPR 1) mindestens noch eine Bilddatei für DPR 2 bereitgestellt wird[65]. (Wird diese Datei nicht bereitgestellt, ist das Verhalten der Applikation undefiniert.)

- Die im Feld 4 referenzierten Bilddateien werden standardmäßig im selben Verzeichnis wie die OAP–Daten abgelegt (s. 3.4). Bei Bedarf können sie jedoch auch in einem Unterverzeichnis davon abgelegt werden. In diesem Fall ist dem Dateinamen ein entsprechender relativer Pfad voranzustellen, wobei der Schrägstrich ('/'=U+002F) als Trennzeichen der einzelnen Pfadkomponenten zu verwenden ist.

- Das Bildformat der in Feld `File` angegebenen Datei muss JPEG, PNG oder SVG sein.

  Bilder im JPEG-Format müssen gemäß der Spezifikation für das „JPEG File Interchange Format (JFIF)“[66] angelegt werden und

  - müssen sequenziell (nicht interlaced/progressive) strukturiert sein
  - müssen Huffman-Kodierung (nicht arithmetische Kodierung) nutzen
  - müssen das YCbCr-Farbmodell nutzen (kein Schwarz/Weiß)
  - müssen 8 Bit-Farbkanal nutzen (nicht mehr).

  Bilder im PNG-Format müssen gemäß der „PNG (Portable Network Graphics) Specification, Version 2.2“[67] angelegt werden und

  - müssen sequenziell (nicht interlaced/progressive) strukturiert sein
  - müssen das RGB-Farbmodell nutzen
  - müssen 8 Bit-Farbkanal nutzen (nicht mehr, kein Schwarz/Weiß)
  - optional ist ein 8 Bit Alphakanal für transparente Bilder erlaubt.

  Bilder im SVG-Format müssen gemäß der Spezifikation „Scalable Vector Graphics (SVG) 1.1 (Second Edition)“[68] angelegt werden, wobei keine Animationen und Interaktionen erlaubt sind.

- Die Bilddateien für eine gegebene Image–ID dürfen bei verschiedenen DPR's (Feld 3) unterschiedliche Bildformate verwenden. So ist es z.B. denkbar bzw. sinnvoll, für die normalauflösende Darstellung eine Rastergrafik (JPEG, PNG) zu verwenden, für höhere Auflösungen hingegen SVG.

[63] welche entsprechend mehr Bilddetails enthalten kann
[64] Bei einer Bilddatei mit mehr als 200x200 Pixel wird das Image entsprechend herunter skaliert.
[65] da dies der DPR von aktuell verbreiteten Smartphones und Tablets entspricht
[66] http://www.jpeg.org/public/jfif.pdf
[67] http://libpng.org/pub/png/spec/1.2/png-1.2-pdg.html
[68] https://www.w3.org/TR/SVG11/

4.12 Versionsinformation

Tabellenname: `Version`
Pflichttabelle: ja

Nr.	Name	Key	Typ	Länge	Pflicht	Erklärung
1.	FormatVersion		Char		X	Nummer der verwendeten Formatversion

Anmerkungen:

- Die Tabelle dient zur Angabe von Informationen über das verwendete Format.
  Die Tabelle darf nur einen Eintrag enthalten.
- Die Formatversion ist in der Form `MajorNumber.MinorNumber` gemäß OAP–Spezifikation anzugeben.

A OAP–Ausdrücke

A.1 Allgemeine Festlegungen

Für boolesche und numerische Ausdrücke wird dieselbe Syntax und Semantik verwendet.

Die Werte von allen, von OAP unterstützten Datentypen (s. A.2) können gemäß der OFML Script Object Notation (OSON) formatiert werden. Davon wird u.a. bei der Übergabe von Argumenten an OFML–Methoden im Rahmen von Aktionen vom Typ *MethodCall* Gebrauch gemacht.

Andersherum können die meisten, wenn nicht alle, gemäß OSON formatierten Wertrepräsentationen geparst und mit einem der von OAP unterstützten Datentypen gespeichert werden. Davon wird u.a. bei Rückgabewerten von OFML–Methoden Gebrauch gemacht, die über die Funktion *methodCall()* aufgerufen werden (s. A.4.15).

Jeder OAP–Ausdruck wird in einem definierten Kontext ausgewertet. Bei der Auswertung der Gültigkeitsbedingung eines Interaktors ist z.B. das aktive Planungselement und die ID des Interaktors bekannt (und relevant). Zu diesem Zweck wird dem Kern-Modul der Applikation, der die Auswertung von OAP–Ausdrücken vornimmt, ein sogenannter **Auswertungskontext** bereitgestellt. Dieser beinhaltet eine *Objekttabelle*, in der für definierte Geltungsbereiche (Scopes) die Referenz auf das betreffende Objekt[69] hinterlegt wird, und eine *Wertetabelle*, in der zu definierten Schlüsseln (z.B. Bezeichner von Platzhaltern, s. A.4.17) die entsprechenden Werte hinterlegt werden.

A.2 Unterstützte Datentypen

Jeder Ausdruck oder Teilausdruck liefert einen Wert, der einen der in den folgenden Unterabschnitten aufgeführten Typen hat.

Für jeden Typ werden Angaben zur OSON–Formatierung bzw. zum Parsen einer OSON–Eingabe gemacht[70].

A.2.1 Error

Werte vom Typ *Error* werden zurückgegeben[71], wenn bei der Auswertung eines (Teil-)Ausdrucks ein Fehler aufgetreten ist. Der Wert besteht aus einem Fehlerkode und einer Fehlermeldung.

OSON–Formatierung: `NULL`
OSON–Parsen: nicht zutreffend

A.2.2 Null

Der Typ *Null* hat als einzigen Wert `NULL`.

Er wird in der Regel verwendet, wenn in einem bestimmten Kontext kein Wert verfügbar ist, wie zum Beispiel bei einem nicht existierenden Merkmal beim Merkmalswertzugriff (s. A.4.16).

OSON–Formatierung und -Parsen: `NULL`

[69] interner Begriff: Article Instance Identifier (AIID)

[70] Bei den dabei genannten Literalen sind die Literale aus Part III des OFML Standards gemeint, nicht die OAP–Literale aus A.3.2.

[71] Fehler werten als spezifisch typisierte Werte zurückgegeben, um bei sofortiger Auswertung während der syntaktischen Analyse Fehler einfacher ignorieren zu können, die in nicht relevanten Operanden von Operatoren mit bedingter Auswertung auftreten.

A.2.3 Int

Ein Wert vom Typ *Int* ist eine ganze Zahl im Bereich von jeweils einschließlich -2^{31} bis $2^{31} - 1$.

Arithmetische Operationen mit ganzzahligem Ergebnis, das nicht als Ganzzahl im angegebenen Bereich dargestellt werden kann, geben einen Fehler zurück.

OSON–Formatierung: dezimales Integer-Literal
OSON–Parsen: dezimales, oktales oder hexadezimales Integer-Literal

A.2.4 Float

Ein Wert vom Type *Float* ist eine 64-Bit Gleitkommazahl entsprechend IEEE 754.

`Infinity` und `NaN` werden nicht unterstützt. Operationen mit entsprechendem Ergebnis geben einen Fehler zurück.

OSON–Formatierung und -Parsen: Gleitkomma-Literal

A.2.5 Symbol

Ein Wert vom Typ *Symbol* ist eine Zeichenkette aus einem üblicherweise begrenzten Wertebereich.

OSON–Formatierung: Wenn möglich als Symbol-Literal, andernfalls als Symbol-Konstruktor (`Symbol()`) mit Zeichenketten-Literal als Argument
OSON–Parsen: Symbol-Literal oder Symbol-Konstruktor

A.2.6 String

Ein Wert vom Typ *String* besteht aus einer möglicherweise leeren Folge aus Zeichen.

OSON–Formatierung und -Parsen: Zeichenketten-Literal

A.2.7 Sequence

Ein Wert vom Typ *Sequence* enthält eine geordnete, möglicherweise leere Folge aus Werten.

OSON–Formatierung: Vector
OSON–Parsen: Vector oder List

A.2.8 Name

Ein Wert vom Typ *Name* repräsentiert einen einfachen Namen wie `t`, einen hierarchischen Namen wie `c.e1` oder einen (teilweise oder voll) qualifizierten Namen wie `::ofml::oi::OiPlElement`.

OSON–Formatierung und -Parsen: einfacher Name, hierarchischer Name, qualifizierter Name

A.2.9 Numerische Typen

Die beiden Typen *Int* und *Float* werden als numerische Typen bezeichnet.

A.2.10 Boolesche Typen

Numerische Typen sowie der Typ *String* werden als boolesche Typen bezeichnet.

Wenn ein Operator einen Operand mit booleschem Typ erwartet, so wird der Wert des Operanden als *true* angesehen, wenn er entweder einen numerischen Typ hat und der Wert ungleich Null ist, oder vom Typ *String* ist und der Wert eine nicht leere Zeichenkette ist. Alle anderen Werte boolescher Typen werden als *false* angesehen.

A.3 Lexikalische Struktur

Ein Ausdruck besteht aus einer nicht leeren Folge von *Token*[72].

Leerzeichen sind vor dem ersten, nach dem letzten und zwischen Token erlaubt.

Token werden unterteilt in *Operatoren* (A.3.1), *Literale*[73] (A.3.2) und *Bezeichner* (A.4.19).

Während der lexikalischen Analyse wird, ausgehend von der aktuellen Position, das nächste Token als die längste mögliche Zeichenfolge ermittelt, die ein gültiges Token ergibt. Die lexikalische Analyse schlägt fehl, wenn kein solches Token ermittelt werden kann.

A.3.1 Operatoren

Die folgenden Operatoren sind definiert:

```
?       :       ::      ||      |       &&      &       ^       ==      !=
!       <       <=      <<      >       >=      >>      >>>     +       -
*       /       %       ~       (       )       $       ,
```

A.3.2 Literale

NULL-Literal

Das NULL-Literal besteht aus den vier aufeinander folgenden Buchstaben `NULL`. Es repräsentiert den einzigen Wert des Typs *Null*.

Symbol-Literale

Es existieren zwei Formen von Symbol-Literalen[74]:

- Die Normalform beginnt mit einem `@`–Zeichen (U+0040), unmittelbar gefolgt von einem Buchstaben oder Unterstrich ('_'=U+005F), gefolgt von null oder mehr Buchstaben, Ziffern und/oder Unterstrich, wobei ausschließlich ASCII–Zeichen erlaubt sind[75].
- Die zweite Form besteht aus einem `@`–Zeichen, unmittelbar gefolgt von einem Zeichenketten-Literal (s.u.). Sie kann und sollte ausschließlich dafür genutzt werden, Symbole darzustellen, die sich mit der Normalform nicht darstellen lassen.

Zeichenketten-Literale

Ein Zeichenketten-Literal besteht aus einer in doppelte Anführungszeichen ('"'=U+0022) eingeschlossenen, möglicherweise leeren Folge von Zeichen. Das doppelte Anführungszeichen selbst ist in der Zeichenkette nicht erlaubt. Um dieses und bestimmte Sonderzeichen darzustellen, sind die folgenden, durch den Backslash ('\'=U+005C) eingeleiteten Escape–Sequenzen zu verwenden:

`\a`	U+0007	Klingelzeichen (BEL)
`\b`	U+0008	Rückschritt (BS)
`\t`	U+0009	Horizontal–Tabulator (HT)
`\n`	U+000A	Zeilenwechsel (NL)
`\v`	U+000B	Vertikal–Tabulator (VT)
`\f`	U+000B	Seitenvorschub (FF)
`\r`	U+000D	Wagenrücklauf (CR)
`\"`	U+0022	Anführungszeichen
`\'`	U+0027	Hochkomma
`\\`	U+005C	Backslash
\[0-7]	Oktale Escape–Sequenz	
`\x`[0-9A-Fa-f]	Hexadezimale Escape–Sequenz	

[72] kleinste sinngebende Einheiten
[73] Literale sind definierte Zeichenfolgen zur direkten Darstellung der Werte von Basistypen.
[74] Das `@`–Zeichen ist nicht Bestandteil des Wertes des Symbols.
[75] vergleiche auch Feldtyp *Symbol* in Abschn. 3.2

Eine oktale Escape–Sequenz besteht aus dem Backslash, gefolgt von bis zu drei oktalen Ziffern ('0'..'7'=U+0030..U+0037), die in einen ganzzahligen Wert gewandelt wird.

Eine hexadezimale Escape–Sequenz besteht aus dem Zeichen `\x`, gefolgt von einem oder mehr hexadezimalen Ziffern ('0'..'9'=U+0030..U+0039, 'A'..'F'=U+0041..U+0046, 'a'..'f'=U+0061..U+0066), wobei die längstmögliche Ziffernfolge konsumiert und in einen ganzzahligen Wert gewandelt wird.

Der durch oktale und hexadezimale Escape–Sequenzen bestimmte Wert modulo 256 ist der Kode des repräsentierten Zeichens.

Folgt dem Backslash keines der oben aufgeführten Zeichen, oder folgt `\x` keine hexadezimale Ziffer, so wird der Backslash ignoriert und das ihm folgende Zeichen unverändert übernommen.

Integer-Literale

Bei Integer-Literalen wird zwischen dezimalen, oktalen und hexadezimalen Integer-Literalen unterschieden:

- *Dezimale Integer-Literale* bestehen aus einer Folge dezimaler Ziffern ('0'..'9'=U+0030..U+0039), wobei die erste Ziffer keine Null sein darf, es sei denn, sie ist die einzige Ziffer.
- *Oktale Integer-Literale* bestehen aus einer führenden Null ('0'=U+0030), gefolgt von ein oder mehr oktalen Ziffern ('0'..'7'=U+0030..U+0037).
- *Hexadezimale Integer-Literale* bestehen aus einer führenden Null ('0'=U+0030), gefolgt von 'x' (U+0078) oder 'X' (U+0058), gefolgt von ein oder mehr hexadezimalen Ziffern ('0'..'9'=U+0030..U+0039, 'A'..'F'=U+0041..U+0046, 'a'..'f'=U+0061..U+0066).

Die durch ein Integer-Literal repräsentierte vorzeichenlose Zahl U darf nicht größer als $2^{32} - 1$ sein. Der vorzeichenbehaftete Wert des Literals I ist gleich U falls $0 \leq U < -2^{31}$, andernfalls gilt $I = U - 2^{32}$.

Gleitkomma-Literale

Gleitkomma-Literale bestehen aus Mantisse und optionalem Exponent.

Die Mantisse besteht aus einer nicht leeren Folge dezimaler Ziffern ('0'..'9'=U+0030..U+0039) sowie maximal einem Dezimalpunkt ('.'=U+002E) an beliebiger Position[76].

Der Exponent besteht aus dem Buchstaben 'e' (U+0065) oder 'E' (U+0045), gefolgt von einem optionalem Vorzeichen '+' (U+002B) oder '-' (U+002D), gefolgt von einer nicht leeren Folge dezimaler Ziffern (s.o.).

Mantisse und Exponent werden generell als Dezimalzahlen interpretiert.

Dezimalpunkt oder Exponent können fehlen, nicht aber beide.

Ein Gleitkomma-Literal, dessen Betrag nach Konvertierung in die interne binäre Darstellung und Rundung auf 53 (binäre) Stellen größer oder gleich 2^{1024} ist, führt zu einem Fehler.

A.4 Syntax von Ausdrücken

A.4.1 Rangfolge und Assoziativität von Operatoren

Die *Präzedenz* (Priorität, Rang) von Operatoren steuert, in welcher Reihenfolge die entsprechenden Operationen in Ausdrücken ausgeführt werden, wenn nicht explizit durch Klammerung eine andere Reihenfolge vorgegeben wurde.

Für Operatoren mit gleicher Präzedenz steuert die *Assoziativität* die Reihenfolge der Auswertung der Operationen. Die meisten Operatoren sind links-assoziativ, so dass *A op B op C* als *(A op B) op C* ausgewertet wird.

[76] einschließlich vor der ersten oder nach der letzten Ziffer

Die folgende Tabelle gibt einen Überblick über Präzedenz und Assoziativität von Operatoren in OAP[77].

Präzedenz	Operator	Beschreibung	Assoziativität
1	`::`	Geltungsbereich für Zugriffe auf Merkmalswerte	
	`$`	Platzhalter	
	`()`	Funktionsaufruf	
2	`+ -`	unäres Plus und Minus	rechts
	`~ !`	bitweise und logische Negation	
3	`* / %`	Multiplikation, Division und Modulo	links
4	`+ -`	Addition und Subtraktion	links
5	`<< >> >>>`	Verschiebeoperationen	links
6	`< > <= >=`	relative Vergleichsoperationen	links
7	`== !=`	Vergleichsoperationen	links
8	`&`	bitweises UND	links
9	`^`	bitweises exklusives ODER	links
10	`|`	bitweises ODER	links
11	`&&`	logisches UND	links
12	`||`	logisches ODER	links
13	`? :`	bedingte Auswertung	rechts
14	`,`	Separator von Ausdrücken in Argumentlisten	links

Teilausdrücke werden generell von links nach rechts ausgewertet[78].

A.4.2 Ausdrücke

Expression:
UnaryExpression
ConditionalExpression

Die Auswertung eines Ausdrucks bzw. Teilausdrucks liefert einen Wert eines der in Abschnitt A.2 beschriebenen Typen.

Wenn ein oder mehrere Teilausdrücke eines Ausdrucks einen Fehler zurückliefern, so wird, wenn nicht explizit anders spezifiziert, der laut Auswertungsreihenfolge zuerst aufgetretene Fehler zurückgegeben. Ob nach Auftreten eines Fehlers in einem Teilausdruck noch weitere Teilausdrücke ausgewertet werden oder nicht, ist nicht spezifiziert.

Wenn in den folgenden Abschnitten gefordert wird, dass eine Bedingung erfüllt sein muss, und die Bedingung nicht erfüllt ist, dann gibt der entsprechende Ausdruck einen Fehler zurück.

A.4.3 Bedingte Auswertung

ConditionalExpression:
LogicalOrExpression
LogicalOrExpression `?` *Expression* `:` *ConditionalExpression*

Der linke Ausdruck muss einen booleschen Wert liefern. Ist der Wert *true*, so wird das Ergebnis des mittleren Ausdrucks zurückgegeben, andernfalls das Ergebnis des rechten Ausdrucks.

Die Implementierung verhält sich dabei so, als ob der jeweils andere Ausdruck nicht ausgewertet werden würde[79].

[77] Die Tabelle hat lediglich informativen Charakter. Die verbindliche Definition von Präzedenz und Assoziativität erfolgt durch die Syntaxregeln in den folgenden Abschnitten.

[78] Mit der aktuell unterstützten Funktionalität ist die Reihenfolge der Auswertung von untergeordneter Bedeutung, da alle Operatoren keine sichtbaren Nebenwirkungen haben und für OFML–Funktionen, die durch *methodCall()* aufgerufen werden, ebenfalls gefordert ist, dass sie keine Nebenwirkung haben dürfen (s. A.4.15).

[79] Implementierungen können den jeweils anderen Teilausdruck spekulativ auswerten, so lange gesichert ist, dass eventuell auftretende Fehler ignoriert werden und keine Nebenwirkungen auftreten, oder ggf. aufgetretene Nebenwirkungen rückgängig gemacht werden, wenn sich die Auswertung des anderen Teilausdrucks zur Ermittlung des Ergebnisses der Ausdruckes als unnötig herausstellen sollte.

A.4.4 Logischer Or–Operator

LogicalOrExpression:
LogicalAndExpression
LogicalOrExpression `||` *LogicalAndExpression*

Der linke Ausdruck muss einen booleschen Wert liefern. Ist der Wert *true*, so ist das Ergebnis `1`. Andernfalls wird der rechte Ausdruck ausgewertet und muss ebenfalls einen boolschen Wert liefern. Ist dieser *true*, so ist das Ergebnis `1`, andernfalls `0`.

Für den Fall, dass der linke Ausdruck *true* ergibt, verhält sich die Implementierung so, als ob der rechte Ausdruck nicht ausgewertet werden würde[80].

A.4.5 Logischer And–Operator

LogicalAndExpression:
BitwiseOrExpression
LogicalAndExpression `&&` *BitwiseOrExpression*

Der linke Ausdruck muss einen booleschen Wert liefern. Ist der Wert *false*, so ist das Ergebnis `0`. Andernfalls wird der rechte Ausdruck ausgewertet und muss ebenfalls einen boolschen Wert liefern. Ist dieser *false*, so ist das Ergebnis `0`, andernfalls `1`.

Für den Fall, dass der linke Ausdruck *false* ergibt, verhält sich die Implementierung so, als ob der rechte Ausdruck nicht ausgewertet werden würde[81].

A.4.6 Bitweise Verknüpfungen

BitwiseOrExpression:
BitwiseExclusiveOrExpression
BitwiseOrExpression `|` *BitwiseExclusiveOrExpression*
BitwiseExclusiveOrExpression:
BitwiseAndExpression
BitwiseExclusiveOrExpression `^` *BitwiseAndExpression*
BitwiseAndExpression:
EqualityExpression
BitwiseAndExpression `&` *EqualityExpression*

Der linke und der rechte Operand müssen vom Typ *Int* sein. Beide Operanden werden dem Operator entsprechend bitweise miteinander verknüpft. Das Ergebnis ist wieder vom Typ *Int*.

A.4.7 Operatoren zum Test auf Gleichheit

EqualityExpression:
RelationalExpression
EqualityExpression `==` *RelationalExpression*
EqualityExpression `!=` *RelationalExpression*

Beim Test auf Gleichheit werden die folgenden Regeln in der angegebenen Reihenfolge angewandt:

- Ist einer der beiden Operanden vom Typ *Null*, so werden beide Operanden als gleich angesehen, wenn beide Operanden vom Typ *Null* sind. Andernfalls sind sie ungleich.

[80] siehe Fußnote oben zur bedingten Auswertung
[81] siehe Fußnote oben zur bedingten Auswertung

- Sind beide Operanden vom Typ *String*, *Symbol* oder *Name*, so werden sie als gleich angesehen, wenn ihre Werte Zeichen für Zeichen übereinstimmen.
- Haben beide Operanden einen numerischen Typ, so werden sie als gleich angesehen, wenn beide den gleichen numerischen Wert haben.
- Sind beide Operanden vom Typ *Sequence*, so liefert der Vergleich einen Fehler, wenn eine der beiden Sequenzen einen Wert vom Type *Error* enthält. Andernfalls werden die Sequenzen als ungleich angesehen, wenn ihre Länge nicht übereinstimmt. Andernfalls findet ein paarweiser Vergleich der Elemente beider Sequenzen statt. Tritt dabei ein Fehler auf, so liefert auch der Vergleich der beiden Sequenzen einen Fehler. Andernfalls werden die Sequenzen als gleich angesehen, wenn alle Elemente paarweise gleich sind.
- Andernfalls liefert der Vergleich einen Fehler.

Das Ergebnis des Operators `==` ist `1`, wenn beide Operanden gleich sind, und `0`, wenn sie ungleich sind.

Das Ergebnis des Operators `!=` ist `1`, wenn beide Operanden ungleich sind, und `0`, wenn sie gleich sind.

A.4.8 Relationale Vergleichsoperatoren

RelationalExpression:
ShiftExpression
RelationalExpression `<` *ShiftExpression*
RelationalExpression `<=` *ShiftExpression*
RelationalExpression `>=` *ShiftExpression*
RelationalExpression `>` *ShiftExpression*

Beim relativen Vergleich zweier Werte werden die folgenden Regeln in der angegebenen Reihenfolge angewandt:

- Sind beide Operanden vom Typ *String*, so werden beide Zeichenketten Zeichen für Zeichen miteinander verglichen, bis ein Unterschied gefunden wird oder das Ende der kürzeren Zeichenkette erreicht wurde. Im Falle eines Unterschieds ist das Ergebnis des Vergleichs der Zeichenketten gleich dem Ergebnis des Vergleichs der sich unterscheidenden Zeichen, wobei das Zeichen mit dem kleineren Zeichencode als kleiner angesehen wird. Andernfalls wird die kürzere Zeichenkette als kleiner angesehen.
- Haben beide Operanden einen numerischen Typ, so ist der Operand kleiner, der den kleineren numerischen Wert hat.
- Andernfalls liefert der Vergleich einen Fehler.

Das Ergebnis des Operators `<` ist `1`, wenn der linke Operand kleiner als der rechte Operand ist.

Das Ergebnis des Operators `<=` ist `1`, wenn der linke Operand kleiner als oder gleich dem rechten Operand ist.

Das Ergebnis des Operators `>=` ist `1`, wenn der linke Operand größer als oder gleich dem rechten Operand ist.

Das Ergebnis des Operators `>` ist `1`, wenn der linke Operand größer als der rechte Operand ist.

Andernfalls ist das Ergebnis der Operatoren `0`.

A.4.9 Verschiebeoperatoren

ShiftExpression:
AdditiveExpression
ShiftExpression `<<` *AdditiveExpression*
ShiftExpression `>>` *AdditiveExpression*
ShiftExpression `>>>` *AdditiveExpression*

Der linke und der rechte Operand müssen vom Typ *Int* sein. Die niederwertigen fünf Bit des rechten Operanden werden als vorzeichenlose Ganzzahl betrachtet und liefern die Anzahl der Bits, im Folgenden n, um die der linke Operand verschoben werden soll.

Der Operator `<<` verschiebt den linken Operanden um n Bits nach links, wobei die höchstwertigen n Bits des Operanden verworfen und die niederwertigsten n Bits des Ergebnisses auf `0` gesetzt werden.

Der Operator `>>` verschiebt den linken Operanden um n Bits nach rechts, wobei das höchstwertige Bit des Operanden in die höchstwertigen n Bits des Ergebnisses kopiert wird und die niederwertigsten n Bits des Operanden verworfen werden.

Der Operator `>>>` verschiebt den linken Operanden um n Bits nach rechts, wobei die niederwertigsten n Bits des Operanden verworfen und die höchstwertigen n Bits des Ergebnisses auf `0` gesetzt werden.

A.4.10 Binäre arithmetische Operatoren

AdditiveExpression:
    MultiplicativeExpression
    AdditiveExpression `+` *MultiplicativeExpression*
    AdditiveExpression `-` *MultiplicativeExpression*
MultiplicativeExpression:
    UnaryExpression
    MultiplicativeExpression `*` *UnaryExpression*
    MultiplicativeExpression `/` *UnaryExpression*
    MultiplicativeExpression `%` *UnaryExpression*

Beide Operanden müssen einen numerischen Typ haben. Der Operator `+` kann darüber hinaus mit zwei Operanden vom Typ *String* verwendet werden.

Wenn beide Operanden einen numerischen Typ haben und mindestens ein Operand vom Typ *Float* ist, wird der andere Operand ggf. nach *Float* konvertiert und die Berechnung in *Float* durchgeführt. Das Ergebnis ist dann ebenfalls vom Typ *Float*. Andernfalls hat das Ergebnis den gleichen Typ wie beide Operanden.

Wenn der Operator `+` auf Operanden vom Typ *String* angewandt wird, so ist das Ergebnis die Verkettung von linken und rechtem Operanden, in dieser Reihenfolge.

Das Ergebnis von Division durch Null oder Modulo Null ist immer ein Fehler, auch wenn die linke Seite des Operators Null ist.

Die Operation $a \ \% \ b$ berechnet das Ergebnis von $a - n * b$, wobei n das in Richtung Null gerundete Ergebnis von a/b ist. Das Ergebnis von $-2^{31} \ \% -1$ ist `0`.

A.4.11 Unäre arithmetische Operatoren

UnaryExpression:
    `+` *UnaryExpression*
    `-` *UnaryExpression*

Wenn der Operand der unären Operatoren `+` und `-` keinen numerischen Typ hat, dann geben diese Operatoren einen Fehler zurück. Andernfalls hat das Ergebnis den gleichen Typ und den gleichen absoluten Wert wie der Operand. Im Falle des Operators `+` hat das Ergebnis das gleiche Vorzeichen wie der Operand. Im Falle des Operators `-` hat das Ergebnis das entgegengesetzte Vorzeichen des Operanden[82].

[82] Einzige Ausnahme ist die Negation von `0`, da es in der Zweierkomplementdarstellung keine Unterscheidung zwischen `+0` und `-0` gibt.

A.4.12 Bitweise und logische Negationsoperatoren

UnaryExpression:
`~` *UnaryExpression*
`!` *UnaryExpression*

Das Ergebnis beider Operatoren ist vom Typ *Int*.

Im Falle des Operators `~` muss der Operand vom Typ *Int* sein. Das Ergebnis ist die bitweise Negation des Operanden.

Im Falle des Operators `!` muss der Operand einen booleschen Typ haben. Das Ergebnis ist `1`, wenn der Operand als *true* angesehen wird, andernfalls `0`.

A.4.13 Primäre Ausdrücke

UnaryExpression:
PrimaryExpression
PrimaryExpression:
`(` *Expression* `)`
FunctionCall
PropertyReference
Placeholder
Literal
`[` $ExpressionList_{opt}$ `]`
ExpressionList:
Expression
ExpressionList `,` *Expression*

A.4.14 Funktionsaufruf

FunctionCall:
Identifier `(` $ExpressionList_{opt}$ `)`

Ein Funktionsaufruf besteht aus dem Namen einer vordefinierten Funktion gefolgt von einer in runde Klammern eingeschlossenen, möglicherweise leeren Liste von durch Komma getrennten Argumenten.

Funktionsaufrufe können verschachtelt werden, d.h., die Argumentliste kann ihrerseits wieder Aufrufe beliebiger Funktionen enthalten.

Wenn bei der Auswertung der Argumente einer Funtion ein Fehler auftrat[83] oder keine Funktion mit dem angegebenen Bezeichner existiert, dann schlägt der Funktionsaufruf fehl.

Aktuell vordefinierte Funktionen sind die im Anhang B beschriebenen Funktionen und die im nächsten Abschnitt beschriebene Funktion *methodCall()*.

A.4.15 Ausführung von MethodCall–Aktionen

Die Funktion *methodCall(*actionId*)* kann verwendet werden, um Funktionsaufrufe durchzuführen, die mittels MethodCall–Aktionen definiert sind. Das Argument *actionId* ist ein beliebiger Ausdruck, der als Ergebnis eine Zeichenkette liefern muss, die als ID der entsprechenden MethodCall–Aktion interpretiert wird.

[83] d.h., eines der Argumente vom Typ *Error* ist

Die Funktion gibt einen Fehler zurück, wenn eine der folgenden Bedingungen erfüllt ist:

- Die Funktion hat nicht genau ein Argument vom Typ *String*.
- Die maximale Anzahl[84] verschachtelter Aufrufe von MethodCall–Aktionen wurde erreicht[85]. Beim Zählen der Tiefe der Verschachtelung wird der Name der Aktion nicht berücksichtigt.
- Im Auswertungskontext des Ausdrucks, der den Aufruf von *methodCall()* enthält, ist das Objekt `SELF` nicht definiert.
- Bei der Abfrage der Daten der MethodCall–Aktion ist ein Fehler aufgetreten, der Typ der Aktion ist nicht *MethodCall* oder die Daten der Aktion sind fehlerhaft.
- Die Ausführungsbedingung der Aktion (falls angegeben) ist nicht eindeutig erfüllt.
- Im Fall einer Instanz–Methode:
  - Bei der Auswertung einer Objektdefinition oder bei der Erzeugung der entsprechenden OFML–Instanz ist ein Fehler aufgetreten.
  - Nach Auswertung aller Objektdefinitionen wurde nicht genau ein Objekt ermittelt.
  - Die Überprüfung des für die MethodCall–Aktion definierten Kontextes (Klasse, Schnittstelle) ist für das ermittelte Objekt fehlgeschlagen.
- Bei der Auswertung der durch die MethodCall–Aktion definierten Argumente unter Benutzung des aktuellen Auswertungskontexts ist ein Fehler aufgetreten.
- Der Aufruf der OFML-Methode ist fehlgeschlagen.
- Das von der OFML-Methode zurückgegebene Ergebnis konnte nicht in einen OAP-Datentyp konvertiert werden.

OFML–Methoden, die durch die Funktion *methodCall()* aufgerufen werden, dürfen keine sichtbaren Nebenwirkungen haben, da es aus Performance-Gründen nicht möglich ist, nach jedem Aufruf von *methodCall()* die in verschiedenen Teilen der Anwendung gespeicherten Informationen über OFML–Instanzen[86] zu aktualisieren.

A.4.16 Zugriff auf Merkmalswerte

PropertyReference:
    PropertyName
    Scope `::` *PropertyName*
PropertyName:
    Identifier
Scope:
    Identifier

Ein Merkmalswertzugriff ohne angegebenem Scope entspricht einem Zugriff mit Scope `SELF` (s.a.u.).

Bei einem Merkmalswertzugriff wird der Bezeichner für den Scope in der Objekttabelle des Auswertungskontextes (s.a. A.1) nachgeschlagen, um den entsprechenden *Article Instance Identifier* (AIID) zu ermitteln. Dieser wird anschließend genutzt, um von der Anwendung den Wert des angegebenen Merkmals der referenzierten Artikelinstanz abzufragen[87].

Wenn in der Objekttabelle kein Eintrag für den angegebenen Scope gefunden wird, das referenzierte Objekt über kein Merkmal mit dem angegebenen Namen verfügt, oder der Wert des Merkmals nicht in einen OAP-Wert konvertiert werden kann, dann liefert der Merkmalswertzugriff als Ergebnis `NULL`. Unterstützte Typen von Merkmalswerten sind *Symbol*, *String*, *Null*, *Int* und *Float*.

[84] In der aktuellen Implementierung (Oktober 2017) ist der Wert auf 10 gesetzt.

[85] Verschachtelte MethodCall–Aktionen können dann auftreten, wenn die Ausführungsbedingung einer MethodCall–Aktion selber die Funktion *methodCall()* verwendet.

[86] wie z.B. Artikelnummern, Variantencodes, Artikel- und Variantentexte

[87] Tatsächlich wird von der aktuellen Implementierung (Oktober 2017) der gesamte PropVarCode von der Anwendung abgefragt und zur eventuellen erneuten Verwendung im Auswertungskontext gespeichert, oder aber ein dort gespeicherter PropVarCode direkt verwendet. Der Merkmalswert wird aus dem PropVarCode extrahiert.

Aktuell sind folgende *Scopes* definiert:

SELF Referenziert das aktive Objekt (entspricht der Objektkategorie *Self*, s. 4.9).

PARENT Referenziert den unmittelbar übergeordneten Artikel des aktiven Objekts (entspricht der Objektkategorie *ParentArticle*, s. 4.9).

TOP Referenziert den in der Hierarchie am höchsten stehenden übergeordneten Artikel des aktiven Objekts (entspricht der Objektkategorie *TopArticle*, s. 4.9).

A.4.17 Platzhalter

Placeholder:
`$` *Identifier*

Bei einer Platzhalterersetzung wird der als Platzhalter angegebene Bezeichner in der Wertetabelle des Auswertungskontextes (s.a. A.1) nachgeschlagen. Wenn ein Eintrag gefunden wurde, dann wird der entsprechende Wert zurückgegeben.

Andernfalls wird der als Platzhalter angegebene Bezeichner in der Objekttabelle des Auswertungskontext nachgeschlagen. Wird auch hier kein Eintrag gefunden, dann ist das Ergebnis der Platzhalterersetzung `NULL`. Andernfalls wird für den in der Objekttabelle gefundenen *Article Instance Identifier* (AIID) die entsprechende OFML–Instanz ermittelt bzw., sollte sie aktuell nicht existieren, neu erzeugt. Ergebnis der Platzhalterersetzung ist dann ein Wert vom Typ *Name*, der den Namen der OFML–Instanz repräsentiert.

Aktuell sind folgende *Platzhalter* definiert:

INTERACTOR
Dieser Platzhalter wird durch den Bezeichner des Interaktors ersetzt, zu dem aktuell Informationen ermittelt werden bzw. an den die aktuell ausgeführte Aktion gebunden ist.

SELF Dieser Platzhalter wird durch den Namen der OFML–Instanz des aktiven Objekts ersetzt (s.a. Scope `SELF` in A.4.16).

A.4.18 Literale

Literal:
SymbolLiteral
StringLiteral
IntegerLiteral
FloatingPointLiteral
NullLiteral

Die Syntax von Literalen ist in Abschnitt A.3.2 beschrieben.

A.4.19 Bezeichner

Ein Bezeichner (*Identifier*) besteht aus einer nicht leeren Folge aus Buchstaben, Ziffern und/oder Unterstrich, wobei das erste Zeichen keine Ziffer sein darf, und ausschließlich ASCII–Zeichen verwendet werden dürfen[88].

[88] entspricht dem Feldtyp *Symbol* in den OAP–Tabellen (s. 3.2)

B Funktionen

B.1 Mathematische Funktionen

Alle Argumente müssen einen numerischen Typ haben (s. A.2.9). Argumente vom Typ *Int* werden vor der Berechnung der Funktion in den Typ *Float* konvertiert. Der Wert der Argumente muss im angegebenen Bereich liegen[89]. Wenn kein Fehler zurückgegeben wird, ist der Rückgabewert bei allen Funktionen vom Typ *Float*.

B.1.1 Trigonometrische Funktionen

acos(x) Berechnet den Arkuskosinus von x für $-1 \leq x \leq 1$.

acosh(x) Berechnet den Areakosinus hyperbolicus von x für $1 \leq x < +\infty$.

asin(x) Berechnet den Arkussinus von x für $-1 \leq x \leq 1$.

asinh(x) Berechnet den Areasinus hyperbolicus von x für $-\infty < x < +\infty$.

atan(x) Berechnet den Arkustangens von x für $-\infty < x < +\infty$. Das Ergebnis liegt im Intervall $[-\pi/2, +\pi/2]$.

atan2(y,x) Berechnet den Arkustangens von y/x für $-\infty < x < +\infty$ und $-\infty < y < +\infty$, wobei die Vorzeichen beider Argumente genutzt werden, um den Quadranten des Rückgabewertes zu bestimmen. Das Ergebnis liegt im Intervall $[-\pi, +\pi]$.

Generell gilt, dass das Ergebnis das Vorzeichen von y hat.

Für positive x ist der Betrag des Ergebnisses kleiner $\pi/2$.

Für negative x ist der Betrag des Ergebnisses größer $\pi/2$.

Im Falle von x gleich Null gilt:

- Wenn y ungleich Null ist, dann ist das Ergebnis $\pm\pi$.
- Wenn x +0.0 ist und y $\pm$0.0 ist, dann ist das Ergebnis $\pm$0.0.
- Wenn x -0.0 ist und y $\pm$0.0 ist, dann ist das Ergebnis $\pm\pi$.

atanh(x) Berechnet den Areatangens hyperbolicus von x für $-1 < x < 1$.

cos(x) Berechnet den Kosinus von x für $-\infty < x < +\infty$.

cosh(x) Berechnet den Kosinus hyperbolicus von x für $-\infty < x < +\infty$.

sin(x) Berechnet den Sinus von x für $-\infty < x < +\infty$.

sinh(x) Berechnet den Sinus hyperbolicus von x für $-\infty < x < +\infty$.

tan(x) Berechnet den Tangens von x für $-\infty < x < +\infty$[90].

tanh(x) Berechnet den Tangens hyperbolicus von x für $-\infty < x < +\infty$.

[89] Auch wenn alle Argumente im angegebenen Bereich liegen, kann ein Fehler zurückgegeben werden, wenn das Ergebnis den Wertebereich von *Float* überschreitet. Unterschreitungen des Wertebereiches (Werte mit sehr kleinem Betrag) führen zu keinem Fehler. Stattdessen wird Null zurückgegeben.

[90] Theoretisch ist die Tangens-Funktion für $\pi/2 + n * \pi$ mit ganzzahligem n nicht definiert. Da π jedoch nicht exakt dargestellt werden kann, sollte das in der Praxis kein Problem sein. So ist z.B. das Ergebnis von $tan(atan2(1,0))$ `1.63312393531954e+16`.

B.1.2 Potenz-, Exponential- und logarithmische Funktionen

exp(x) Berechnet den Wert der Exponentialfunktion e^x für $-\infty < x < +\infty$.

log(x) Berechnet den natürlichen Logarithmus von x für $0 < x < +\infty$.

log10(x) Berechnet den dekadischen Logarithmus von x für $0 < x < +\infty$.

logb(x) Berechnet den in Richtung negativ unendlich gerundeten binären Logarithmus von x für $\infty < x < +\infty$[91].

pow(x,y) Berechnet den Wert von x^y für $-\infty < x < +\infty$ und $-\infty < y < +\infty$.

Wenn x negativ ist, muss y ein ganzzahliger Wert sein.

Wenn x `0` ist, darf y nicht negativ sein.

Das Ergebnis ist `1.0`, wenn sowohl x und y `0` sind.

scalb(x,y) Berechnet $x * 2^n$, wobei n gleich y gerundet in Richtung Null ist, für $-\infty < x < +\infty$ und $-\infty < y < +\infty$.

sqrt(x) Berechnet die Quadratwurzel von x für $0 \leq x < +\infty$.

B.1.3 Rundung, Absolutwert und Rest

ceil(x) Berechnet für $-\infty < x < +\infty$ den kleinsten ganzzahligen Wert, der nicht kleiner als x ist.

fabs(x) Berechnet den absoluten Betrag von x für $-\infty < x < +\infty$.

floor(x) Berechnet für $-\infty < x < +\infty$ den größten ganzzahligen Wert, der nicht größer als x ist.

fmod(x,y) Berechnet den Rest der Gleitkommadivision x/y für $-\infty < x < +\infty$ und $-\infty < y < +\infty$ sowie $y \neq 0.0$.

Dies entspricht der Operation $x\%y$, mit dem Unterschied, dass das Ergebnis immer vom Typ *Float* ist.

remainder(x,y) Berechnet den Rest der Division x/y für $-\infty < x < +\infty$ und $-\infty < y < +\infty$ sowie $y \neq 0.0$.

Das Ergebnis ist $x - n * y$, wobei n das Ergebnis von x/y gerundet zur nächsten ganzen Zahl ist. Wenn der absolute Wert von $x - n * y$ `0.5` ist, dann wird n so gewählt, dass es gerade ist.

[91] Das Ergebnis ist gleich dem Wert des Exponenten der internen Gleitkommarepräsentation konvertiert nach *Float*.

B.2 Funktionen zur Typkonvertierung

Alle Konvertierungsfunktionen existieren in zwei Varianten.

Die erste Variante hat ein Argument, das den zu konvertierenden Wert spezifiziert.
Sie liefert als Ergebnis einen Fehler (Wert vom Typ *Error*), wenn die Konvertierung nicht erfolgreich war.

Die zweite Variante hat zwei Argumente. Das erste Argument spezifiziert den zu konvertierenden Wert, das zweite einen Standardwert, der als Rückgabewert der Konvertierungsfunktion verwendet wird, wenn die Konvertierung nicht erfolgreich war. Der Typ des Standardwerts ist keinen Einschränkungen unterworfen. Es ist nicht definiert, ob bei erfolgreicher Konvertierung der Ausdruck im Argument für den Standardwert ausgewertet wird oder nicht.

B.2.1 Konvertierung nach *Int*

int(value)
int(value, default)

Die Funktion konvertiert den Wert des im Argument *value* übergebenen Ausdrucks in einen Wert vom Typ *Int*.

Die folgenden Konvertierungen werden unterstützt:

- von *Int* nach *Int*:

  Der Rückgabewert der Konvertierungsfunktion ist gleich dem Wert des Arguments *value*.

  Diese Konvertierung ist immer erfolgreich.

- von *Float* nach *Int*:

  Im ersten Schritt der Konvertierung werden eventuelle Nachkommastellen vom Wert des Arguments *value* abgeschnitten (Rundung in Richtung Null).

  Die Konvertierungfunktion liefert den so ermittelten Wert als *Int*, wenn er als solcher darstellbar ist ($-2^{31} \leq I < 2^{31}$). Andernfalls ist die Konvertierung nicht erfolgreich.

- von *String* nach *Int*:

  Bei dem Wert des Arguments *value* muss es sich um eine aus den folgenden Komponenten bestehende Zeichenkette handeln: optionale führende Leerzeichen, ein optionales negatives Vorzeichen ('-'=U+002D), ein Integer-Literal wie in Anhang A.3.2 beschrieben, optionale abschließende Leerzeichen. Andernfalls ist die Konvertierung nicht erfolgreich.

  Ohne Vorzeichen ist der Rückgabewert der Konvertierungsfunktion gleich dem Wert des Literals, andernfalls gleich dem negierten Wert des Literals (s. A.4.11).

B.2.2 Konvertierung nach *Float*

float(value)
float(value, default)

Die Funktion konvertiert den Wert des im Argument *value* übergebenen Ausdrucks in einen Wert vom Typ *Float*.

Die folgenden Konvertierungen werden unterstützt:

- von *Int* nach *Float*:

  Der Rückgabewert der Konvertierungsfunktion ist eine Gleitkommazahl mit dem gleichen Wert wie das Argument *value*.

  Diese Konvertierung ist immer erfolgreich.

- von *Float* nach *Float*:

  Der Rückgabewert der Konvertierungsfunktion ist gleich dem Wert des Arguments *value*.

  Diese Konvertierung ist immer erfolgreich.

- von *String* nach *Float*:

  Bei dem Wert des Arguments *value* muss es sich um eine aus den folgenden Komponenten bestehende Zeichenkette handeln: optionale führende Leerzeichen, ein optionales negatives Vorzeichen ('-'=U+002D), ein Gleitkomma-Literal wie in Anhang A.3.2 beschrieben, optionale abschließende Leerzeichen. Andernfalls ist die Konvertierung nicht erfolgreich.

  Ohne Vorzeichen ist der Rückgabewert der Konvertierungsfunktion gleich dem Wert des Literals, andernfalls gleich dem negierten Wert des Literals (s. A.4.11).

B.2.3 Konvertierung nach *Symbol*

symbol(value)
symbol(value, default)

Die Funktion konvertiert den Wert des im Argument *value* übergebenen Ausdrucks in einen Wert vom Typ *Symbol*.

Die folgenden Konvertierungen werden unterstützt:

- von *Symbol* nach *Symbol*:

  Der Rückgabewert der Konvertierungsfunktion ist gleich dem Wert des Arguments *value*.

  Diese Konvertierung ist immer erfolgreich.

- von *String* nach *Symbol*:

  Die Konvertierungsfunktion liefert einen Wert vom Typ *Symbol*, der exakt die gleiche Zeichenfolge repräsentiert wie die zu konvertierende Zeichenkette.

  Diese Konvertierung ist immer erfolgreich.

  > Hinweis:
  > Es findet keine Sonderbehandlung für das Zeichen '`@`' (U+0040) am Anfang der Zeichenkette statt. Der Ausdruck `symbol("@FOO")` liefert also ein Symbol, das nach Formatierung gemäß der OFML Script Object Notation (OSON) die Zeichenkette `"Symbol(\"@FOO\")"` liefert (anstatt `"@FOO"`).

B.2.4 Konvertierung nach *String*

string(value)
string(value, default)

Die Funktion konvertiert den Wert des im Argument *value* übergebenen Ausdrucks in einen Wert vom Typ *String*.

Die folgenden Konvertierungen werden unterstützt:

- von *Int* nach *String*:

  Der Rückgabewert der Konvertierungsfunktion besteht aus einer Zeichenfolge, die den Regeln für dezimale Integer-Literale entspricht (siehe Anhang A.3.2) und den gleichen Wert wie das Argument *value* repräsentiert.

  Diese Konvertierung ist immer erfolgreich.

- von *Float* nach *String*:

  Der Rückgabewert der Konvertierungsfunktion besteht aus einer Zeichenfolge, die den Regeln für Gleitkomma-Literale entspricht (s. A.3.2). Abgesehen davon ist die exakte Repräsentation bis auf die folgenden Ausnahmen unspezifiziert:

- Die Zeichenkette enthält Dezimalpunkt oder Exponent, oder beide.
- Die Konvertierung erfolgt mit einer Genauigkeit von mindestens fünfzehn Dezimalstellen in der Mantisse.
- Die Konvertierung produziert ab der zweiten Nachkommastelle der Mantisse keine nicht signifikanten Nullen.

Die Konvertierung von Gleitkommazahlen nach Zeichenketten ist immer erfolgreich.

> Hinweis:
> Die Konvertierung ist nicht exakt. Es ist nicht sichergestellt, dass der Ausdruck `float(string(x)) == x` wahr ist.

- von *Symbol* nach *String*:

  Die Konvertierungsfunktion liefert einen Wert vom Typ *String*, der exakt die gleiche Zeichenfolge repräsentiert wie das zu konvertierende Symbol.

  Diese Konvertierung ist immer erfolgreich.

  > Hinweis:
  > Der Wert des Ausdrucks `string(@FOO)` ist `"FOO"` (nicht `"@FOO"`).

- von *String* nach *String*:

  Der Rückgabewert der Konvertierungsfunktion ist gleich dem Wert des Arguments *value*.

  Diese Konvertierung ist immer erfolgreich.

C Allgemeine Regeln zur Verarbeitung von Property Variant Codes

Um eine konsistente Verarbeitung von Property Variant Codes in allen Applikationen sicherzustellen und für zukünftige Erweiterungen, z.B. mehrwertige Merkmale, vorbereitet zu sein, sollen Property Variant Codes einheitlich wie folgt verarbeitet werden:

1. Aufspalten des PropVarCodes in einzelne, durch Semikolon (';'=U+003B) getrennte *Felder*.

   Dazu wird der PropVarCode von links nach rechts analysiert, wobei Backslash ('\'=U+005C) und doppelte Anführungszeichen ('"'=U+0022) besonders behandelt werden: Im Falle eines doppelten Anführungszeichen werden alle folgenden Zeichen bis zum einschließlich nächsten doppelten Anführungszeichen oder dem Ende des PropVarCode ignoriert, wobei ein Backslash die besondere Bedeutung des nachfolgenden Zeichens, also von Backslash und doppelten Anführungszeichen, aufhebt.

   Jedes Feld besteht also aus der längsten möglichen Folge von Zeichen, die dem erweiterten regulären Ausdruck `([^;"]|"([^\"]|(\\.))*("|$))*` genügt.

2. Zerlegung der Felder in *Schlüssel* und *Wert* sowie Entfernen aller *ungültigen* Felder.

   Ein gültiges Feld besteht aus einem gültigem Schlüssel unmittelbar gefolgt von einem Gleichheitszeichen ('='=U+003D) sowie einem möglicherweise ungültigen Wert, wobei dieser aus allen unmittelbar auf das Gleichheitszeichen folgenden Zeichen besteht. Gültige Schlüssel genügen dem im Abschn. 3.2 definierten Feldtyp *Symbol*.

   Gültige Werte sind alle im Anhang A beschriebenen Literale mit optionalem vorangestellten Vorzeichen ('+'=U+002B oder '-'=U+002D) vor Integer- und Gleitkomma–Literalen.

3. Die Behandlung von Feldern mit ungültigem Wert ist kontextspezifisch:
   - Beim Zugriff auf ein Merkmal mit ungültigem Wert in OAP–Ausdrücken ist das Ergebnis `NULL` (s.a. A.4.16).
   - Beim Vergleich zweier PropVarCodes ergibt der Test auf Gleichheit zweier Werte immer falsch, sobald mindestens einer der beiden Werte ungültig ist, auch dann, wenn beide Werte durch die gleiche Zeichenfolge repräsentiert werden (s.a. 4.1.3).

4. Das Verhalten im Falle von PropVarCodes mit zwei oder mehr Feldern mit gleichem Schlüssel aber unterschiedlichen Werten (nach Umwandlung in die interne Darstellung) ist abhängig von der jeweiligen Implementierung. Bei gleichen Werten soll sich die Implementierung aber so verhalten, als wenn nur ein Eintrag vorhanden wäre.

D Änderungshistorie

OAP 1.5, 1. überarbeitete Fassung:

- Ergänzungen und Präzisierungen zur Sichtbarkeit von Interaktoren (Abschn. 4.6) sowie zur Gültigkeit von Aktionen inkl. Beschreibung des (bereits unterstützten) Dummy-Aktionstyps *NoAction* (Abschnitte 4.7 und 4.8.1).

OAP 1.5:

- Neue Interaktor–Symboltypen *Electrification*, *Lighting*, *RotateNY* und *RotatePY* (Abschn. 4.6).
- Bei Aktionen vom Typ *DimChange* können nun auch symbolische Properties vom Typ `YS` (Werte vom Typ *String*) verwendet werden (Abschn. 4.8.4).
- Alle Referenzen auf den in Version 1.2 abgekündigten Aktionstyp *PropEdit* entfernt.

OAP 1.4, 1. überarbeitete Fassung:

- Korrektur und Präzisierung zur Verwendung der Objektkategorie *MethodCall* (Abschn. 4.9).

OAP 1.4:

- Neuer Interaktor–Symboltyp *Attention* (Abschn. 4.6).
- Neuer Aktionstyp *Message* (Abschn. 4.7).
- Präzisierung zur Verwendung von Anwendungsbedingungen für Properties und Klassen bei einer Aktion von Typ *PropEdit2* (Abschn. 4.8.3).
- Bei Aktionen vom Typ *DimChange* können nun auch Properties verwendet werden, die eine symbolische Auswahlliste repräsentieren, wenn die Mapping-Methode *symbolicPropValue2Float()* implementiert ist (Abschn. 4.8.4).
- Neuer Abschnitt zur Beschreibung der Parametertabelle für den neuen Aktionstyp *Message* (Abschn. 4.8.7).
- Präzisierung zur Verwendung von Escape-Sequenzen in Texten (Abschn. 4.10).

OAP 1.3:

- Korrektur hinsichtlich möglicher Nachfolgeaktionen nach Aktionen des Typs *PropEdit2* (Abschn. 4.7).
- Präzisierung zur Behandlung des Status bei einer Aktion des Typs *PropEdit2* mit nur einer gültigen Property (Abschn. 4.8.3).
- Beim Merkmalszugriff werden nun die Scopes PARENT und TOP unterstützt (Anhang A.4.16).
- Der Platzhalter $INTERACTOR wird nun generell unterstützt[92] (Anhang A.4.17).

OAP 1.2:

- Die Beschreibungen von noch nicht unterstützten Tabellen und Funktionalitäten wurden entfernt oder grau unterlegt.
- Korrekturen und Präzisierungen zum Interaktor–Konzept (Abschn. 2.1).
- Das Feld *NeedsPlanMode* in der Tabelle `Interactor` ist *abgekündigt* (Abschn. 4.6).
- Neue Interaktor–Symboltypen *StartDimChange* und *Video* (Abschn. 4.6).
- Der Aktionstyp *PropEdit* wird *abgekündigt*. Stattdessen sollte der neue Aktionstyp *PropEdit2* verwendet werden (Abschn. 4.7).
- Neuer Abschnitt zur Beschreibung der Parametertabellen für den neuen Aktionstyp *PropEdit2* (Abschn. 4.8.3).

[92] nicht nur in Gültigkeitsbedingungen von Interaktoren

- Präzisierung zu den Materialbildern für Property-Werte im `PropEdit2`–Dialog (Abschn. 4.8.3).
- Im Feld *Dimension* kann in der Tabelle `DimChange` (Abschn. 4.8.4) nun auch festgelegt werden, in welcher Achsenrichtung eine Änderung erlaubt ist. Des Weiteren wurden die Felder *Separate* und *ThirdDim* eingeführt.
  Wird ein bestehendes Projekt nach OAP 1.2 portiert, muss also eine ggf. vorhandene Tabelle `DimChange` angepasst werden!
- In der Tabelle `CreateObj` wurde der PosRotMode *AttachAreas* entfernt (Abschn. 4.8.5).
- Neuer Abschnitt für die Parametertabelle `ExtMedia` (Abschn. 4.8.8).

OAP 1.1:

- Neuer Interaktor–Symboltyp *OnOff* (Abschn. 4.6).